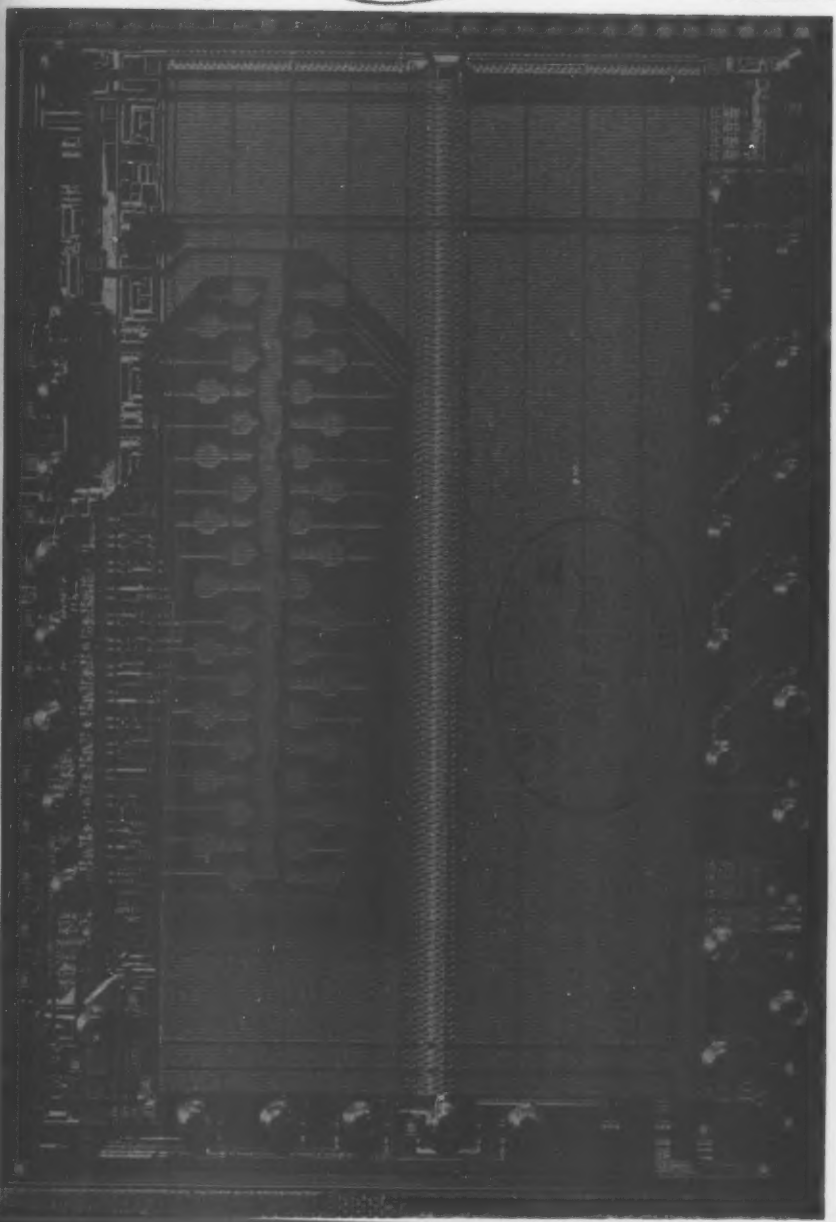
Concepts

The Journal of
Defense Systems
Acquisition Management

Autumn 1982
Volume 5
Number 4

GOVERNMENT DOCUMENT



Cover

Our cover illustration is a close-up of IBM's 72K-bit dynamic random-access memory chip, one of the largest-capacity chips currently in volume production anywhere. The actual size of the chip is indicated by the rectangular block in front of the word "cover" above. (Photo courtesy of IBM Corporation).

Concepts (ISSN 0279-6759) is published quarterly by the Defense Systems Management College, Fort Belvoir, Va. 22060, and is intended to be a vehicle for the transmission of information on policies, trends, events, and current thinking affecting program management and defense systems acquisition.

Statements of fact or opinion appearing in *Concepts* are solely those of the authors and are not necessarily endorsed by the Department of Defense or the Defense Systems Management College. Unless copyrighted, articles may be reprinted. When reprinting, please credit the author and *Concepts*, and forward two copies of the reprinted material to the Editor.

Manuscripts and other correspondence are welcome and should be addressed to the Editor. Inquiries concerning proposed articles or distribution may be made by phone at (703) 664-5082 or AUTOVON 354-5082. Second class rate entered at Fort Belvoir, Va.

POSTMASTER: Send address changes to Editor, *Concepts*, Defense Systems Management College, Fort Belvoir, Va. 22060.

Concepts

The Journal of
Defense Systems
Acquisition Management

Autumn 1982
Volume 5
Number 4

EDITORIAL BOARD

Clarence G. Carlson
Vice President
Hughes Aircraft Co.

John W. Gouy
Vice President, Contracts and Pricing
McDonnell Aircraft Co.

Robert M. Powell
LMSC Vice President
Assistant General Manager, Space Systems Division
Lockheed Missiles and Space Co., Inc.

Major General Edward M. Browne, USA
Program Manager, Advanced Attack Helicopter
U.S. Army Aviation Research and Development Command

Lieutenant Colonel Ronald L. Charbonneau, USA (Ret.)
Director, Corporate Project Management
International Paper Company

Captain Clifford A. Rose, USN
Project Manager, Undersea Surveillance
Naval Electronic Systems Command

Colonel Arthur J. Wilson III, USAF
Deputy for Mission Support Systems
Air Force Systems Command

Commander Charles A. Vinroot, USN
Technical Director
Cruiser Destroyer Acquisition Project
Naval Sea Systems Command

Major Lester L. Lyles, USAF
F-16 Multinational Staged Improvement Program Project Manager
Air Force Systems Command

DEFENSE SYSTEMS MANAGEMENT COLLEGE

Brigadier General Benjamin J. Pellegrini, USA
Commandant

Colonel Dirk H. Lueders, USA
Deputy Commandant

Colonel G. Dana Brabson, USAF
Dean, Department of
Research and Information

CONCEPTS

Robert Wayne Moore
Editor

Catherine M. Clark
Assistant Editor

Susan G. Pollock
Deborah L. McVee
Dorothy L. Reago
Editorial Assistants

Greg Caruth
Cover Design

Fred Hughes
Layout and Chart Design

Concepts

The Journal of
Defense Systems
Acquisition Management

Autumn 1982
Volume 5
Number 4

9 **DOD Policy for Acquisition of Embedded Computer Resources**
H. Mark Grove

Embedded computer resources (ECR) costs are expected to grow at an exponential rate in the decade of the '80s, yet deficiencies exist in the way these resources and the associated costs are managed. This paper addresses some of those management difficulties as well as some of the proposed solutions. It specifically discusses the limitation on high-order languages and instruction set architectures as a means of control.

37 **Military Computers in Transition: Standards and Strategy**
Louise Giovane Becker

The Department of Defense has a need for computer support of "mission-critical" operation, that is, computers that are integral parts of defensive and offensive weapon systems as well as those other systems associated with critical military operations. To ensure efficient and effective utilization of these resources, DOD has promoted standardization of both equipment and software. This paper discusses DOD initiatives in achieving such standardization and some of the observations that have been made on the standardization strategy.

48 **Suggestions for DOD Management of Computer Software**
AIA/EIA White Paper

This paper, jointly sponsored by the Aerospace Industries Association (AIA) and Electronic Industries Association (EIA) Government Division, addresses the problems the Department of Defense faces in managing embedded computer resources in an environment of rapidly moving technology. It surveys industry's perceptions of DOD's current software management techniques, and offers a number of specific recommendations intended to help DOD manage ECR more efficiently.

73 **Today's Risks in Software Development—Can They Be Significantly Reduced?**
Major General Jerry Max Bunyard, USA
James Mike Coward

There is significant risk associated with the development of software for complex operations, which is borne out by the record of software cost overruns, schedule slippages, and software products that do not meet expectations. The authors examine the risks often encountered in software development with the objective of focusing on lessons learned during the past 15 years and developing recommendations that can result in significantly reducing these risks.

95 **A Software Quality Framework** — *Dr. Emanuel R. Baker, Dr. Matthew J. Fisher*

It is the authors' contention that the ability of programmers and software designers to produce correct software lies at the heart of the difficulties in software management. Because it is a "human endeavor," software development is an "art" that, according to the authors, lacks discipline. They outline a quality program designed to instill such a discipline in software development and therefore improve the process and the product.

108 **Software Quality Assurance** — *James A. Dobbins, Robert D. Buck*

Because of the growing requirement for even more reliable software, the discipline of software quality assurance is receiving increased attention in defense systems acquisition. This paper presents the standards for software quality assurance programs and discusses a program of implementation that offers both management control of the process and early customer visibility.

119 **Improving the Combat Direction System Acquisition Cycle** — *Lieutenant Commander Marvin J. Langston, USN*

Building superior command and control systems today requires, in the author's judgment, timely acquisition of the latest hardware using well-defined software in an appropriately engineered system architecture. The author discusses the requirement for those involved in embedded computer resources systems acquisition to develop methods that will intelligently integrate state-of-the-art hardware with well-designed software. He proposes the use of advanced development models built with commercial equipment to benefit the process and shorten the acquisition cycle.

125 **Evolution of Teleprocessing** — *Charles K. Watt*

The information society is rapidly becoming a reality, with technological advances blurring the traditional lines separating communications devices and processors. This move into "teleprocessing," defined as the combined use of communications facilities and data processing equipment, is expected to accelerate in the years ahead. This paper is intended to identify the major technology drivers, systems relationships, and trends expected to have an impact on teleprocessing services during the next decade.

141 **Defense Computers and Software—What's Ahead for AI?** — *Earl C. Joseph*

One of the most exciting possibilities for computers lies in their use as "expert" systems capable not only of storing knowledge but also capable of using inference procedures to solve problems. The author discusses the current state of artificial intelligence and looks to a future when AI systems will be an essential creative management tool.

148 **Software Safety Management** — *David A. Davis, Dr. Nancy G. Leveson*

Software safety is becoming increasingly important as computers begin to control more and more of the safety-critical components of weapon systems. The authors argue that this increased reliance on software, coupled with the fact that error-free code cannot be quaranteed, has made it more important than ever that management give software safety the closest attention. They discuss safety program planning and describe a promising software safety analysis tool.

155 **Software Logistics: A Sleeping Giant** *Paul J. McIlvaine*

The increasing commitment on the part of the Department of Defense to develop military systems with embedded computer resources is having a significant impact on logistics support. The author discusses this impact and some of the logistics concerns that must be addressed if true software integrated logistics support is to be a future reality.

162 **A Management Technique for Software Acquisition**
Major Howard W. Wendt, USAF
Michael W. Evans

Although the form of software management problems may be unique, the specific management techniques, practices, and procedures required to develop, integrate, and demonstrate software are, according to the authors, based on good engineering management principles. This paper discusses an approach to the government's responsibility in managing software development, and presents management tools that can be applied during the pre-award, post-award, and production phases of software development.

177 **The Evolving Nature of the C^3 Systems Acquisition Process**
Robert B. Doane

The author suggests that the efforts of the Air Force to field, within a reasonable period of time, operationally effective command, control, and communications (C^3) systems has been hindered by problems stemming, to a great extent, from the unique nature of C^3 systems and the rapidly evolving potential provided by the computer explosion. He outlines some lessons learned from previous C^3 development efforts, and recommends changes intended to make future acquisitions more efficient.

191 **Joint Service Software Policy and Standards**
Lieutenant Colonel Casper H. Klucas, USA
Major Larry A. Fry, USAF
Lieutenant Commander John W. Barnes, USN
Dr. Matthew Fisher

In 1979, the Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resources Management recognized the opportunity for the services to standardize at least a part of the policies, procedures, regulations, and standards related to computer software acquisition and support. At a software workshop held in 1979, a subgroup of the Policy Coordinating Group concluded that joint-service policy and standards should be developed. In this paper the authors describe the draft joint-service policy regulation that has been developed.

202 **Software Needs Top-Down Management** *David R. Brown, Jr.*

In the author's view, the remarkable advances that have been made in computer hardware and programming technology during the past 30 years have not been matched by advances in software management practices. He examines what may be the major causes of software management problems, including the prevalent "myths" surrounding software management, and offers suggestions for improving that management through a top-down approach.

from the editor . . .

Many predicted it would happen, but few could have guessed the day would come so soon when computers would touch almost every facet of our lives. But that day has come, and with it has come a new way of looking at the world, a vision enhanced by the almost limitless potential of man working with his most fascinating, most awe-inspiring creation—the computer.

The most exciting aspect of the computer is not what we have harnessed these machines to do, but what they may make it possible for us to do in the future. Possibilities we could not have dreamt of a decade ago are now within our grasp, or soon will be. We have no choice but to pursue these possibilities to realization, and by so doing to improve the health, safety, and general welfare of people everywhere.

Possibilities must also be pursued in the defense arena so that technological advances in data handling and interpretation can be quickly translated into enhanced national security.

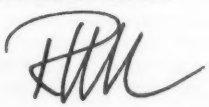
Some of those possibilities are already being pursued, with the use of embedded computer resources (ECR) becoming more prevalent, and with sophisticated military systems becoming more dependent on well-designed and reliable software. But this pursuit has brought us face to face with complexities and obstacles that are every bit as unique as the possibilities we so enjoy pondering. Those obstcles must be overcome and the difficulties resolved if the computer age is to be a benefit to the American defense effort rather than a detriment.

This issue of *Concepts* is devoted to some of the concerns that must be addressed as we try to obtain those benefits and accelerate the melding of computer resources and military systems. Some of these concerns revolve around policy matters; some involve technology. We have tried to cover the most significant issues in defense computer and software acquisition management in an effort to clarify the problems and perhaps offer suggestions for dealing with them.

Acknowledgments

We would like to express our appreciation to two people in particular who were instrumental in helping us put together this special issue of *Concepts*. The first is Jerry L. Raveling, Systems Manager with the Technical Services Division of Sperry Univac and Chairman of the G-33 Computer Resources, Data and Configuration Management Committee of the Electronic Industries Association (EIA). Mr. Raveling worked with us and with members of the G-33 Committee and EIA for almost a year and helped generate some of the articles included in this issue. We are also grateful to Lieutenant Colonel Kenneth E. Nidiffer, USAF, an instructor in the Technical Management Department at the Defense Systems Management College and Course Director of DSMC's Management of Software Acquisition Course. Colonel Nidiffer helped us plan the content of this issue and also advised us in the final selection of material.

Without the able and enthusiastic support of these two men, this issue would not have been possible. We extend to them our most sincere thanks.



Foreword to Special Edition

H. Mark Grove
Assistant Deputy Secretary of Defense
for Research and Advanced Technology

The DOD embedded computer resources (ECR) environment today reflects a history of problems the military departments have encountered in their expanding use of computers in systems that must survive and interoperate in wartime environments. There has been uncontrolled proliferation of hardware and software, low interoperability and interchangeability, and maintenance and logistics difficulties. The result is uncomfortably high life-cycle costs and a lack of satisfaction with the final product.

Estimates by independent sources indicate that DOD's investment in embedded computer hardware and software will increase from approximately $4.1 billion in 1980 to more than $30 billion in 1990. The nature and source of the high cost have also been changing. Technological advances in the computer field have, for example, shifted the systems life-cycle cost from hardware to software. Hardware life-cycle cost now represents only a small percentage of the total system cost. In the case of software, operational life-cycle cost exceeds by far development cost. It is estimated that software maintenance cost is approximately 70 percent of the life-cycle cost for most weapon systems.

Technological advances driving software cost in electronic and computer systems include breakthroughs in microelectronics, which have made possible the creation of smaller and faster machines well-suited for weapon systems applications. Solid-state developments have provided the capability for expanded use of firmware, especially with microprocessors. The use of distributed systems has been gaining in popularity. Very large-scale integration (VLSI) and very high-speed integrated circuit (VHSIC) technology is being developed with a great degree of success. All of these developments will have long-range logistics and operations impacts.

Development of testing tools and facilities for hardware and software have become critical to the successful acquisition of computer systems and must be considered in the initial stages of system development. The same is true for configuration control, quality control, maintainability, reliability, availability, and the rest of the "ility" litany.

New approaches and philosophies have to be developed and applied to the management of computer resources in DOD. Software development, application, and support cannot be treated in the same traditional manner as hardware. It must be treated as a dynamic system, keeping in mind that its integration with applicable software is a continuous process. The same is true for logistic support applications, software quality control, safety, security, and other nonperformance attributes.

The Department of Defense has initiated a number of actions designed to improve the acquisition, management, and control of computer resources. Policies are being developed with the aim of reducing time to deployment, life-cycle cost,

training, equipment proliferation, unnecessary duplication, and complexity of system changes while enhancing maintainability, interoperability, survivability, and hardware/software interchangeability and transportability.

One of the most important recent initiatives in DOD was the establishment, by the Under Secretary of Defense for Research and Engineering (USDR&E) in August 1981, of a Defense Science Board Task Force on Embedded Computer Resources Acquisition and Management. The Task Force was directed to "review, evaluate and make recommendations concerning the acquisition, management and utilization of digital computers and associated technology to support the military mission of the Department of Defense." One of the major issues considered by the Task Force was the proposed standardization of a limited number of instruction set architectures (ISAs) as delineated in a proposed DOD Instruction 5000.5X. The goal of this policy is to control the proliferation of software in DOD's weapon systems by limiting the number of instruction set architectures from more than 700 to 6. The Task Force also addressed the high-order language (HOL) standardization issue and the policy limiting the number of HOLs to be used by DOD under DOD Instruction 5000.31 from more than 150 to only 7, with the ultimate goal of using Ada* as the common HOL for the broadest spectrum of computer application in mission applications. Many other initiatives are under way since the magnitude of the task on hand is unprecedented due to the complexity of the issues involved. The content of this issue of *Concepts* will provide further insight and hopefully a better understanding of these issues. ||

*"Ada" is a registered trademark of the U.S. Department of Defense.

9 DOD Policy for Acquisition of Embedded Computer Resources

H. Mark Grove

Before we begin a discussion of this subject, we must establish what we mean by "embedded computer resources" (ECR). We must establish not only what it is, but also what it is not. Although those most intimately involved with computer systems and associated software have been active in the field for several years, this matter of definition is still troublesome. Specifically, there must be some differentiation between ECR and the more general-purpose computer resources that are subject to the acquisition process governed by the Brooks Act (Public Law 89-306).

Initially, during the early 1970s, ECR was defined as follows:

> All computer equipment, programs, data, documentation, personnel and supplies integral to a defense system from the design, acquisition or operations and support point of view [DOD Directive 5000.29].
>
> Excludes general-purpose, commercially available automatic data processing equipment used for administrative data processing [DOD Directive 5100.40].

More generally, ECR represent the fourth bullet in the class we now term "mission-critical" computer resources (MCCR) which was established by Section 908 of the FY 1982 Defense Authorization Act. That is, mission-critical computer resources includes automatic data processing equipment or services if the function, operation, or use:

—Involves intelligence activities;
—Involves cryptologic activities related to national security;
—Involves command and control of military forces;
—Involves equipment that is an integral part of a weapon or weapon system, or;
—Is critical to direct fulfillment of military or intelligence missions.

The Defense Authorization Act exempted DOD from provisions of the Brooks Act to provide the Department with more complete control, within the program management or acquisition process, over those resources that are the heart of major weapon systems.

The distinction that results from this newer definition is a step away from an approach that was independent of application and dependent on the source of

---

H. Mark Grove is the Assistant Deputy Under Secretary of Defense for Research and Advanced Technology. He is the Senior DOD policy official for all activities related to the acquisition and management of automatic data processing equipment and services covered in Section 2315 of Title 10, U.S. Code. Mr. Grove is the Chairman of the Management Steering Committee for Embedded Computer Resources, a committee with oversight responsibilities for ECR in DOD. He has previously held DOD-level positions in standardization and support programs and in computer resources and electronics. Mr. Grove holds an E.E. degree from the University of Cincinnati, and an M.Sc. from The Ohio State University.

computer or software toward a definition that is application-oriented and more suitable for operational use. This distinction is drawn because of the criticality in dealing with systems that kill people, destroy property, or affect the collection of intelligence information, or with systems that directly support a mission.

As a further test of the definition, we may ask whether it includes equipment in post-mission analysis, and the answer is not totally clear. The issue could center around whether the system is dedicated full time to that use such that it could not be used for any other shared purpose, or whether security considerations are primordial. Examples of such a system might include automatic test equipment, or avionics integrated shop facilities used for avionics software development. In general, however, if a system is associated with the direct military mission of DOD, or if it will be critical to that mission, the tendency will be to classify it as a mission-critical system. This will reduce acquisition time and ensure that the program manager and others involved in the acquisition of a military system have control over both technical and economic aspects of those resources.

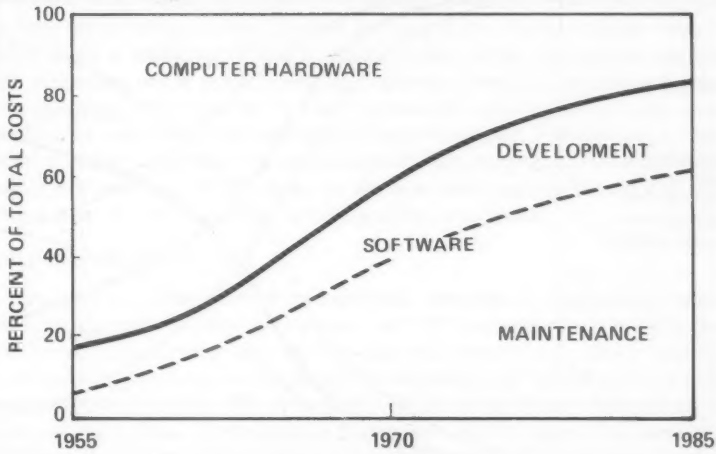
Special Management Processes

Increasingly, special management processes are applied in the embedded computer resources arena, and it is important to realize the necessity for such intensive management. First, software is big business in DOD. In the 1974–75 time period, the DOD annual investment was about $3 billion. At that time, about two-thirds of the programs or systems that relied highly on the use of the computer were only in the development phase, so there was a "bow wave" that could be expected as the system moved along in its life cycle into what has been improperly called "maintenance" of software. Those expectations for increases, which were astounding at that time, have since been exceeded.

A second reason for providing special management attention to ECR is the increasing reliance on embedded systems, both hardware and software. Seven years ago, no one contemplated nor suspected the magnitude of the explosion in microprocessors and microcomputers or the impact of providing full life-cycle support.

The last issue involved in this concern is the serious erosion of the technological lead enjoyed by the United States. The United States, quite properly, has considered itself to be in the forefront in software and computer technology. However, the normal diffusion process functions in business as well as it does in semiconductors, and if the concentration is high, one can expect to observe the flow of that technology at a rather rapid rate to regions of lesser concentrations. That expectation has been borne out in experience. In addition to the normal flow of technology outward, however, some judgmental errors in

FIGURE 1
DOD Computer System Life-Cycle Cost Trends



business or investment strategy were made. First, the "industry first—defense follow" strategy was followed. That strategy invokes the theory of letting industry do the bulk of work first for commercial applications and then adapting the results to defense applications. That theory doesn't work universally. It hasn't worked here. In addition, software was considered to be "technical data" and therefore could be purchased in the same manner as drawings are acquired, leading to inadequate information for provision of organic support of software.

The life-cycle cost trends in DOD computer systems are also very revealing in the sense of management requirements. Figure 1 is well known, having been rather widely published by Dr. Barry Boehm. Analysis of the surveys shows that the leverage in software cost will not be in development, but in maintenance of software, or support of software in the post-deployment phases as time progresses. An additional inference from Figure 1 is that the cost of hardware has, over the last few years, been decreasing at an ever-increasing rate—one of the few technological areas in which one receives much more for much less over time. The curves displayed show that the original problem attacked from a development standpoint and the research and engineering standpoint will be transferred into the logistic support business rather rapidly, and experience has shown the trends to be correct.

FIGURE 2
Software Impact

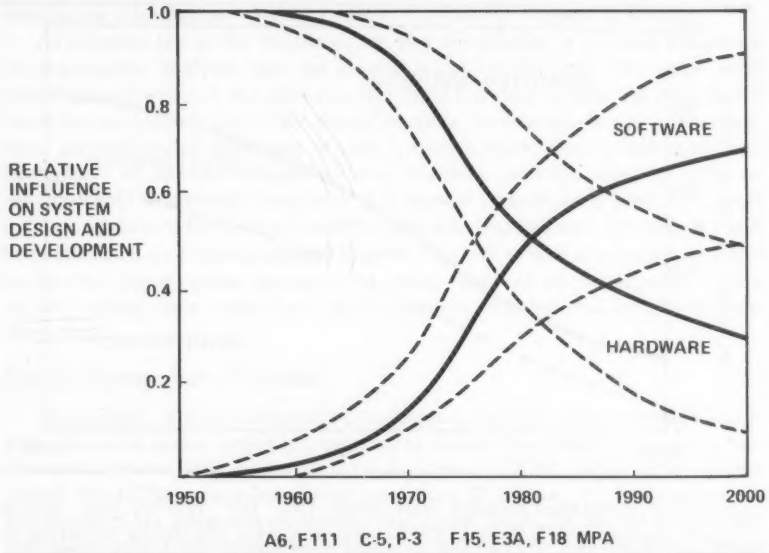


Figure 2 provides another view of that same issue and illustrates that the impact of software on the design of systems is an ever-increasing function. It graphically illustrates an idea that is only now being realized—that software is a system. It is no longer merely a *part* of a major system, but a *system* that performs the integration functions for the systems, whether they are avionics, or missiles, or command and control functions. Integration of system functions is now found in the information domain and, as a result, ever-increasing attention must be given to management to avoid the extension of the software cycle in the normal pattern. This means that the system acquisition strategy must take software seriously into account. In a few recent cases, the software segment was actually a source-selection determinant.

It has been said by some observers that DOD often de-emphasizes the control of software and hardware, bringing software under the same management processes that have been used for many years. Were that the case, then one might forecast the death of modern systems as they are known. Still, there are indica-

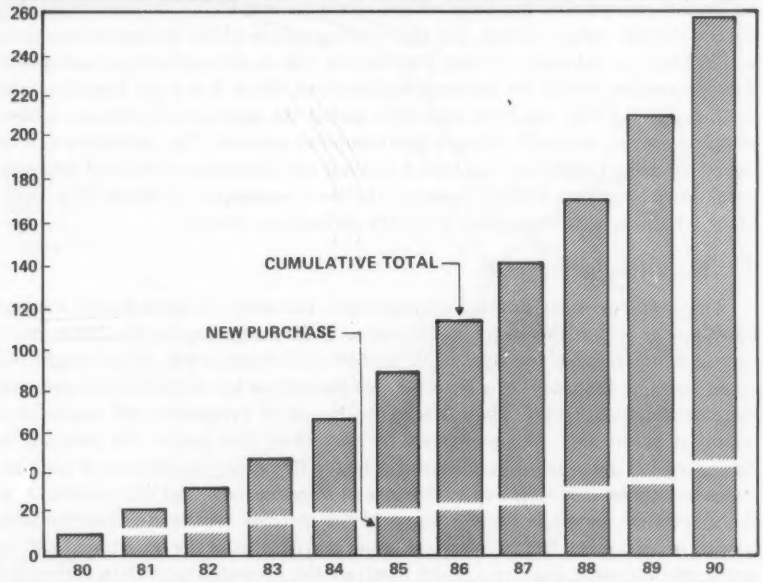
tions that this claim may be a reality and therefore another basis upon which to insist upon special management processes. The Electronic Industries Association (EIA) performed a study that led to a 10-year forecast. The study concluded that we do not really know the scope of the problem, that DOD has been managing the total dollar value, or cost, but that the magnitude of the computer resources within DOD is unknown. It was pointed out that in the commercial automatic data processing world for business applications, there is a good base for such data, but that DOD has been unable to gather the appropriate data for understanding the real scope of software and computer systems. This deficiency can be traced to earlier judgments that held it was not very important to extend the work breakdown structure (WBS) down to the level necessary to obtain that data. Once a habit is well engrained, it is very difficult to change.

The Budget Bulge for ECR

The need for more pointed management, however, is immediately evident when one considers the various forecasts of DOD budgeting for the 1980s. Such forecasts indicate that the total DOD budget will almost triple, which might not be so startling. However, it is expected that the budget for the electronics area will increase almost fourfold. The increase for the use of computers will approach a factor of seven and, of significance to readers of this paper, the increase in budgeting for software should exceed a factor of eight. The decade of such increases might be characterized as the "age of information," and it is essential that the investment be made wisely. Some of the increase will be an investment in hardware, of course, but increasingly larger percentages of the investment will be applied to software. Figures 3 and 4 illustrate the projected growth in embedded computers and relative allocations of investment, respectively. Figure 3 shows a cumulative inventory of embedded computers by year through the decade, beginning at a number below 10,000 in the 1980-1981 time frame and growing to nearly 260,000 by the end of the decade. The implications on logistic supportability should be immediately apparent when it is realized that if it is a problem now, the problem could be astronomical when the inventory of embedded computers rises by a factor of 25.

Another significant factor, as shown in Figure 4, is the distribution of the added investments previously cited. Although the hardware is increasing rapidly, the percentage of the total dollar attributed to software will increase more rapidly, by a factor of nearly a third, or from 65 percent of the total at the beginning of the decade to 85 percent of the total by the end of the period. Another illustration of this growth is shown in Figure 5. It shows that estimates made several years ago for 1980 were fairly accurate. The flatness of the curve in the first part of the decade is a further projection that was previously estimated. The EIA study that

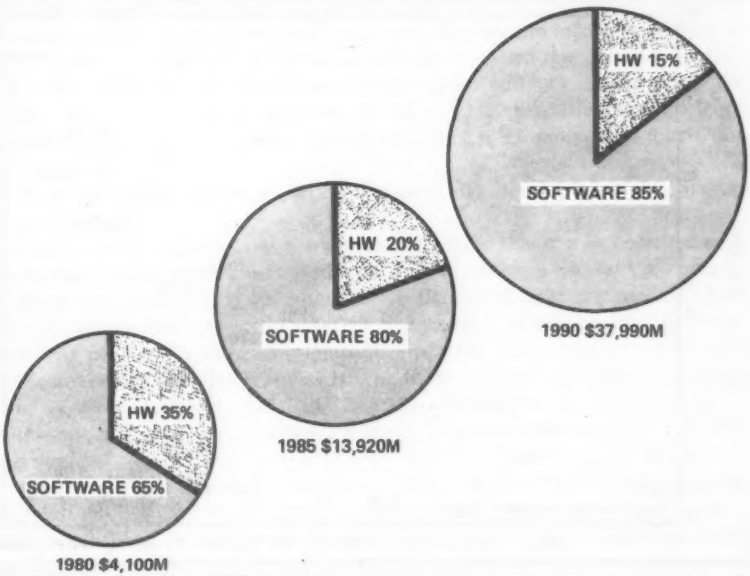
FIGURE 3
DOD Embedded Computers



produced this curve resulted from querying member companies and consulting numerous experts in the field. The study confirms the figures and, more importantly, the shape of the curve and the distribution of investment between hardware and software. The exponential shape of the total investment curve, of which the major part is attributed to software, is staggering when one considers the implications on personnel matters—finding the people who can develop and provide the software, yet maintain it once it is in the field.

Another disturbing element identified in Figure 5 is the absolute dollar number of $38 billion for the year 1990. It is beyond one's imagination how such an investment could be managed, or even assimilated, by DOD. Since this amount is so high, some of the initiatives undertaken over the past few years will require acceleration in order to flatten the trend during the middle of the decade. Regardless of the country's propensity toward increasing the research and development and procurement accounts in the budget, the investment of nearly

FIGURE 4
Embedded Computers Hardware vs. Software



$40 billion in embedded computer resources simply cannot be afforded. This is not a new observation, as indicated in the following quote:

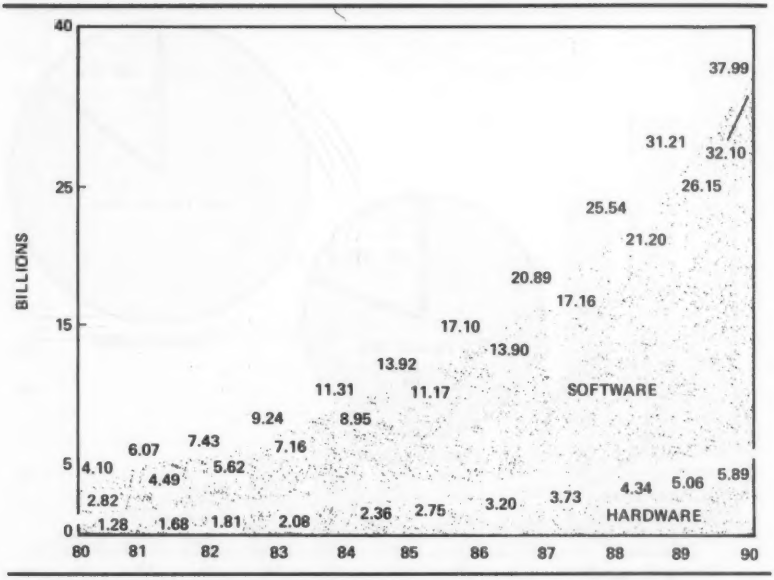
> Large increases in cost with questionable increases in performance can be tolerated only for race horses and fancy women.
>
> Lord Kelvin

To Lord Kelvin also is attributed recognition of the importance of measuring concepts, with the thought that if they cannot be quantified, they are not understood.

Obstacles to Effective ECR Management

The problems that underlie this exploding investment can be summarized as follows:

FIGURE 5
DOD Embedded Computer Market



There is a very large investment in software. The foregoing discussion supports this assertion. This investment tends to cause decision-making based on "sunk funds," which is just as bad for software as for any other business decision.

Support software availability paces applications software. Availability of software support facilities is essential to provide a foundation for the development of application software. It is often more a decision-making parameter than merely a pacing item. A program manager and his prime contractor may have an inefficient facility with which both are comfortable in that they are familiar with the faults and vagaries of the facility. Unfortunately, local decisions are made on this basis, and the resulting design is not necessarily the best for the overall military department and the DOD. We believe that sharing software across systems will gain us more than will continuing to make decisions on individual program bases alone.

Resources are dwindling. The dollar cost is increasing. For every one of the unique architectures and languages that has been used, there has been a concomitant requirement for a special facility. The shortage of people, especially in the software area, is an ever-growing concern. The DOD has experienced difficulty in retaining good people at any level because of the premium industry is willing to pay for those who perform similar jobs, and that situation is expected to persist. It has been estimated that one million programmers will be required within the decade, and with a 9-month gestation period and a 20-year training/experience period before productivity peaks, there is no simple solution to the problem, nor a simple way to plan.

Those problems have a sequence of causes historically, listed in summary form as follows:

Lack of early visibility. Software is not something that can be observed with the normal human senses, as is hardware. It does not have that sort of concrete attribute and, as a result, does not receive that much management visibility. Management cannot see exactly what the programmers are doing and, therefore, even after nearly 30 years in the computer era, most senior managers have very little appreciation for what software is. They have a growing appreciation for the value of computation in the digital world—what it can do both for and to them—but the separation between the hardware and software aspects has been well hidden.

Insufficient management discipline. With little visibility, management discipline has been lacking. Program managers in this area are wont to manage in their own fashion, or to trust a junior staff member or contractor to make most of the decisions about computer resources.

No life-cycle perspective. Without the management discipline cited above, managers are principally motivated to produce the system as nearly on schedule as possible and with the minimum overrun they can anticipate. What is often neglected is consideration of the impact upon downstream costs, since overruns might occur after delivery, in the post-deployment support phase of systems.

Inadequate technology base. Earlier, the diffusion of the U.S. technology lead in the computer software area was cited. A contributing factor was that most of the investment during the early research and development phase of a system development for work in computer and software development is accomplished at a computer-science level which, although academically and theoretically interesting and challenging, involves a 5-year minimum latency between the time the computer scientists release something and the time its significance is felt in systems.

Lack of standardization. This problem arises from the inclination for each system to have its own architecture and the resulting disinterest in sharing lessons that were learned from earlier attempts at similar developments, or sharing engineering approaches for possible joint, or across-the-board use.

Lack of transferability. The lack of a thrust toward standardization may be one of the underlying reasons for lack of the transferability of techniques or lessons learned from one program to another. Another significant cause arises from the "not invented here" syndrome which, although probably denied in most circles, still reigns free.

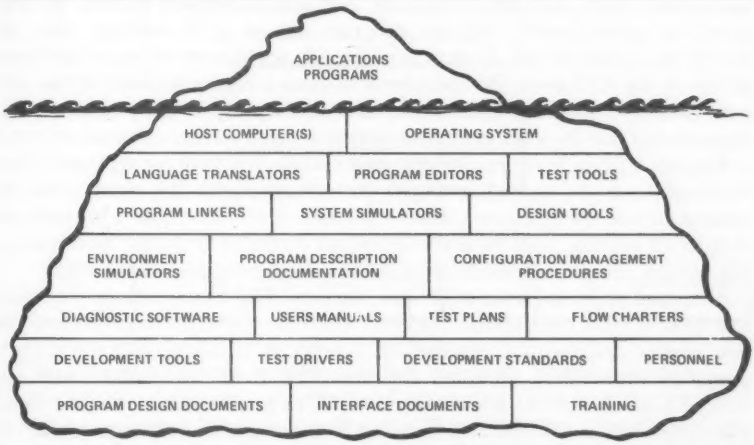
Inappropriate business practices. Elements of this factor have been cited previously; however, only part of the issue is addressed because of the concentration on applications programs in major systems. That part of the "software iceberg" shown in Figure 6 that is involved with applications programs, however, is small in relation to the total. Whether that portion is 9 percent or 10 percent is not known because there is not a common understanding of the foundation of the iceberg below the water line and how much could possibly be saved if that larger bulk of the investment could be shared across systems. An improved understanding of the magnitude of investment represented by the elements portrayed below the water line provides the rationale for most of the management processes that address standardization and the control of the numbers of languages used in the development of software, or the various architectures employed for the hardware.

Policy Initiatives

As a result of the problems affecting embedded computer resources and the dismal forecasts of investment requirements, DOD has undertaken several policy initiatives that are addressed in DOD Directive 5000.29, "Management of Computer Resources in Major Defense Systems." This Directive was issued in April of 1976. The first attack from a policy viewpoint was aimed at improving the understanding of program personnel that computer resources in major subsystems must be considered and that attention must be given throughout the phases of a system life cycle, from development through acquisition and deployment, and especially through the support phase, in which the investment is as much as three times that of the other phases.

A second policy initiative derived from realizing the implications of the software/hardware trade-offs is shown in Figure 5. Analysis showed that little could be gained in pursuing improvement in the area of hardware in a reasonable amount of time. The selected alternative to that approach was to address language issues, which resulted in issuance of DOD Instruction 5000.31 in

FIGURE 6
The Software Iceberg



November of 1976. This instruction listed seven high-order languages that were approved for use in systems. That instruction, coupled with DOD Directive 5000.29, established the policy that higher-order languages, languages that are closer to the problem descriptions and closer to the human side of the interface of the computer, would be used on systems unless it was either technically or economically not feasible to do so. The burden of proof was on the program manager to show that compliance with the policy for his individual system would be inappropriate, or incorrect, rather than on the DOD or the military service to prove the reverse. While experience has not been totally satisfactory in this regard, there are signs of increasing compliance as cost and schedule overruns continue to haunt many of the programs. It should be noted that the ultimate goal from the start is to reduce to a minimum that small set of languages for the development software. To help meet this objective, a program was established that has matured into the Ada® [1] programs to provide a common language.

Automatic test equipment (ATE) language is another concern that was not addressed until 1978, 2 years after the issuance of the earlier policy directives. It has

1. Ada is a registered trademark of the U.S. Department of Defense.

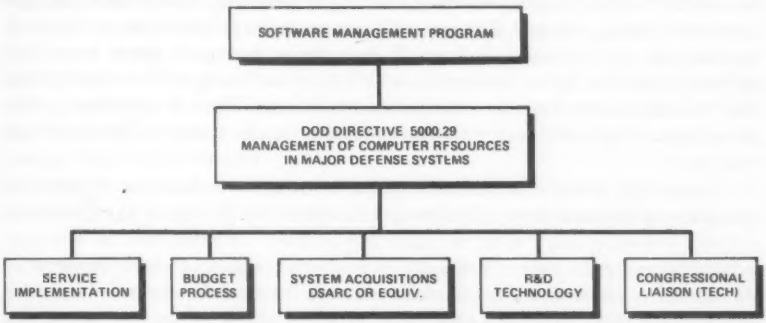
been shown that there probably is more investment in the ATE software than there is applied toward direct military operational systems, confirming the suspicion that the early forecasts were conservative. For example, there are some 70 operational flight programs driving the F-16 aircraft weapon system, as compared to approximately 500 test program sets of ATE software that test something on the aircraft. That ratio of roughly seven to ten times as much emphasis on the ATE as on the operational systems is strong evidence of the software problem. A further aggravation, or compounding, of the problem results from the fact that the support systems were developed with no attention to standardization, leading to the requirement that each facility built for support of line-replaceable units be tailored, having its own characteristics. It is obvious that the massive part of the investment below the "water line" was required for each one of the ATE systems, with the attendant impact on the DOD budget. In defense of the services, however, it should be noted that some progress is being made.

The final policy initiative addressed, again in 1978, was an attempt to reduce to a small number the architectures upon which the software must function. The first attempt described the architectures in terms of hardware, with the military nomenclature. These included the AN/UYK-7 for the Navy, and the AN/UYK-20, AN/AYK-14 and the AN/GYK-12 for the Army. The Air Force was excused from this first attack since it had so many architectures it was impractical to select. A partial explanation for this situation is that the Air Force acquisition process is slightly different from the other two services in that they do not provide government-furnished equipment for most of their systems. They describe their equipment in terms of military standards at the architecture level. When this policy initiative was first suggested, it met opposition from the industry, fearing they would be eliminated from competition by the description in terms of nomenclature and indicating that cost in the hardware area would explode even further. These attempts proved to be false starts in trying to describe the hardware or substrate in terms of other than proprietary design.

Technology Initiatives

Concurrent with the initiatives cited above, some technology initiatives were introduced to bring a functional cut of software computer resources across the applied and exploratory development part of the research and development program. The System Software Management Plan was the resulting tool for implementation of the initiative. This was intended, from a technology viewpoint, to elicit attention by laboratory-level personnel, in whom was vested a technical capability that was considered farther up the learning curve, to all of the life-cycle phases of the system. Traditionally, laboratories in general are reluctant to embrace such an approach.

FIGURE 7



The issue of standards, whether interface standards, language standards, or architecture standards, was addressed and an attempt made to arrive finally at adequate proofing of the support software iceberg before the program manager had to make his decision. Often, the lack of standardization has been excused on the basis that the support software is not "mature," with little consideration given to the fact that the people who use the support software are not necessarily mature either.

These fundamental policy and technology issues were then articulated in a revision of the capstone acquisition directives, DOD Instructions 5000.1 and 5000.2, in December of 1980 and for the first time, the issue of computers was explicitly addressed, not by implication, but by specific reference. In simplified terms, the revised instruction made it clear that computer resources should be acquired in the context of the system, and not in the context of the source.

The Software Management Program

The elements of the Software Management Program are shown in Figure 7. This program and other responsibilities have been assigned to the oversight of a committee. The Management Steering Committee for Embedded Computer Resources is the group that was assigned the responsibility, and DOD Directive 5000.29 provides some insight from an acquisition management and policy standpoint into areas that policy-making officials seldom tread, such as service implementation, the budget process, system acquisitions DSARC or equivalent, R&D

technology and congressional liaison. The directive assumes extra significance in that it is one of the first places that acquisition managers have, by charter, been assigned responsibility to appraise the R&D programs and to have some measure of control over the content and conduct of such programs. It also gave recognition to the need for informal liaison with congressional staffs, a function normally reserved for legislative liaison offices, whose personnel bring very little technical expertise to the interaction with Congress. Many of the congressional staff members are technically well qualified in this era. There is a need for professional interaction and this directive provides at least a modicum of license to meet that need.

Department of Defense Directive 5000.29 encompasses the areas of problems and policy presented above. However, to reiterate for emphasis, the Directive:
—Considers subsystems to be of major importance;
—Establishes requirements for validation and risk analysis, of which elements little has been seen in the documentation that is forwarded concerning systems, although the requirement has existed;
—Imposes a configuration management discipline for software, an area in which mistakes have been made in assuming that the procedures and processes employed in the management of hardware systems can be directly applied to software systems;
—Requires life-cycle planning for software and deliverability of the unique support software that is required if systems are to be organically supported;
—Defines the milestones for software within the DSARC process and establishes attainment criteria for approval;
—Addresses software language standardization and control.

In addition, DOD 5000.29 established a Management Steering Committee (MSC) for ECR, which I chair, to perform oversight functions. The MSC consists of two major segments: (1) an Executive Board consisting of representatives of the secretarial level of the various military departments and DOD agencies, and (2) a working-level Management Advisory Board consisting of experts who provide advice concerning the various proposed policy changes in the computer and software business. All services and most of the agencies are represented on the committee, providing a forum for informal interaction and an environment in which problems can be assessed and acted upon without concern for what one's particular service position might be. Most of the effort in this activity is performed in subpanels, or technical panels. There are five such subpanels, in various states of organization and activity:

—*High Order Language Working Group (HOLWG).* This panel actually continues activity that was initiated before the directives, before DOD realized there

was a significant problem. This activity was the real base from which stemmed the concern over software technology.

—*Research and Development (R&D) Technology Panel.* This group provides and maintains the master plan and performs the oversight functions for individual programs in the military departments. This oversight is not necessarily intended to prevent duplication, but to ensure that any duplication has a maximum productivity to it. The panel meets roughly quarterly, although recently somewhat more frequently. It participates in the apportionment review process in the DOD and has some measure of influence within DOD over the distribution of the meager resources available.

—*Architecture Panel.* Essentially short-lived and now dormant, this panel's activity led to an early conclusion that there need be no particular concern about instruction set architectures.

—*Personnel and Training Panel.* Not fully staffed at this writing, nevertheless this panel's early deliberations were directed toward establishing a charter that led, in some measure, to the design of a course addressing the Management of Software Acquisition at the Defense Systems Management College. As the activity of this panel progresses, it will encompass other activities necessary in a reasonable overview sense as attempts are made to improve the training and education of service and civilian personnel complements in order to alleviate the personnel shortage problem.

One result of the MSC's activity was the identification within the individual services of focal points, as follows:

—Army—ACS-AC
—Navy—NAVMAT TECPO
—Air Force—AFSC/ALR

This appointment of foci has proven most beneficial. The DOD has some 73 laboratories and over 60,000 people assigned, and the ability to reach those activities and personnel through just three contacts is a very practicable arrangement.

Standardization in Languages

I mentioned that DOD Instruction 5000.31 was intended to control the number of higher-order languages in use and to aim at eventually minimizing that number. A revision to 5000.31 is in process, although not formally issued, but it reflects that activity to date has been successful in reducing the original seven approved languages to six:

—Ada
—CMS-2
—JOVIAL (J73)
—COBOL

—FORTRAN
—ATLAS

Ada is the language that has been the ultimate objective over the past 7 to 10 years for achievement of a DOD common language. The redraft of the instruction should be issued soon and it would behoove managers to acquaint themselves with activity in the Ada language world, in the hopes that in about 1983, they can take the first real positions to employ that language. The Army, Air Force, and the Navy, in that order, have established internal policies pending issuance of the formal revision; however, some progress is being made regardless of the bureaucratic lethargy.

To progress toward adoption of Ada as the ultimate common language for the services, a unique approach has been devised. The responsibility has been delegated to a joint program office that operates at the Deputy Under Secretary of Defense level. Only two organizations are afforded that kind of visibility within the Office of the Secretary of Defense (OSD), Ada being one of them. The other is an organization that deals with the very-high-speed integrated circuits program, into which some $320 million will be invested over the next few years. The Ada program is much more conservative and modest than that, with only $5 million per year invested into the program, or a total of some $50 million to reach an implementation phase within the departments. The goal of the Ada program office is to provide a single high-order language for writing software for DOD embedded computer (real-time) applications, a language that is fairly close to the program and close to the hardware. The concentration is on real-time application, and there is no intention to force COBOL, FORTRAN and the other business/scientific languages out the door. The hope is, of course, that Ada will have enough facility and capability to naturally become a language choice in the future; however, that is not the main objective. The objective is to improve that software design and operation efficiency by concentrating scarce resources in a restricted area, on one language.

Let me digress for an interesting note concerning Ada, since I have used the term several times. Ada is not an acronym, as are most shortened program titles in DOD. Rather, it is an "eponym." The language is named Ada after the world's first recognized programmer. Augusta Ada Byron, Countess of Lovelace (1815-1852), and the daughter of Lord Byron. Ada did her work for Babbage's analytical engine which, although never built, was the forerunner of modern computers. A further example of the fetish of the program personnel was their numbering of the military standard describing the language as MIL-STD-1815, the 1815 being the year of Ada Byron's birth. Purely by coincidence, MIL-STD-1815 is dated on December 10th, which was Ada's birth date, and was an-

nounced on December 12th, which is the anniversary of the date that Ada's birth was announced in the *Times* of London. Actually, it had been planned to date the MIL-STD some 6 to 12 months earlier, but bureaucracy being as it is, there became a fortuitous opportunity for public relations and advertising.

Studies into Software Processes

Earlier in this paper, the concerns about software processes that were expressed in the 1974 time frame and earlier were articulated. In 1980, the communication, command and control and intelligence (C^3I) community appraised the situation and arrived at a set of conclusions as to the state of the process and the support process at that time. They identified the following:

—Multiplicity of standards;
—Inadequate acquisition planning;
—Lack of good management practices;
—Emphasis on hardware costs rather than on software costs;
—Software acquisition and contract type often mismatched;
—Security requirements rarely considered in estimating costs and schedules;
—Software development is unique;
—Life-cycle cost estimating inadequate;
—Software productivity tools need to be developed;
—Software development needs to be standardized.

Most of those conclusions look embarrassingly familiar. They are similar to those identified over 6 years prior to this study. However, it is also fair to state that the group performing the study for the C^3I people were not terribly well grounded as to the activity in the systems arena. However, that fact only changes the percentages or the weight that one puts on the results, because the problems remain real. While good progress has been made in some of these areas, there are two or three issues that could be emphasized. Software development is, in fact, unique and one should not necessarily use the same processes that are used for hardware; however, the engineering disciplines that have matured on hardware should be transferred over to the extent practicable. Also, life-cycle cost estimating is an issue that has yet to be resolved. The models currently used are poor, and there is very little correlation from one model to another, indicating effort is needed in the technology base. Productivity tools—the compilers, the various data bases and so forth that managers need to do a good job—are yet to be developed in many cases. Also, it has been previously pointed out in this paper that software development needs to be standardized. The Ada program has a segment that is specifically aimed to the programming environment situation, and there is a need to provide more emphasis there, for if slipping to one language, one development

environment, can be achieved, then the software development needs will, perforce, be standardized—if its use can be enforced.

About the same time that the above-mentioned study was reported in June of 1980, the General Accounting Office (GAO) initiated a study that is believed to have been self-generated, rather than one requested by a congressional staff. It may have been engendered by the publicity given to Ada and the military computer family in the Army at that time and the resulting campaign amongst industry to bring some rationality to the situation. The General Accounting Office spent considerable time on the study, some 6 to 12 months, and visited and interviewed numerous experts in both the services and industry. The initial findings were very critical, but GAO afforded the DOD the opportunity to discuss the efforts made in addressing the management process, what had already been accomplished, and what was in the planning stage. After such discussion, GAO's final recommendations fell quite uniformly in line with DOD's position as to what remained to be accomplished. It is believed to be one of the few cases in which the GAO final report[2] was actually a joint exercise between DOD and the GAO. Their recommendations were to:

—Establish Ada as the standard programming language for military applications;
—Establish standard instruction set architectures (ISAs) to be used in various military applications;
—Establish the extent or appropriate level of computer standardization to be achieved beyond ISA;
—Strengthen the decision-making management organization to oversee defense-wide standardization efforts.

The final recommendation is the only one with which DOD disagreed at the time the report was published. The recommendation that the Management Steering Committee be made a decision-making body is counter to the line and staff concept practiced in the DOD. As a result, DOD, in its response to the report, influenced GAO to accept the DOD position against the recommendation, although it remains to be seen whether GAO may have been correct. The Defense Science Board is reviewing several related issues and it will be asked to have an outside body, preferably an industrial body, to reassess the GAO position and recommend an appropriate direction to take.

The Role of Hardware in Software Acquisition

Up to this point, this paper has dealt with software, with just brief excursions

2. GAO Report, LCD-80-69, *The Department of Defense's Standardization Program for Military Computers—A More Unified Effort is Needed*, June 18, 1980.

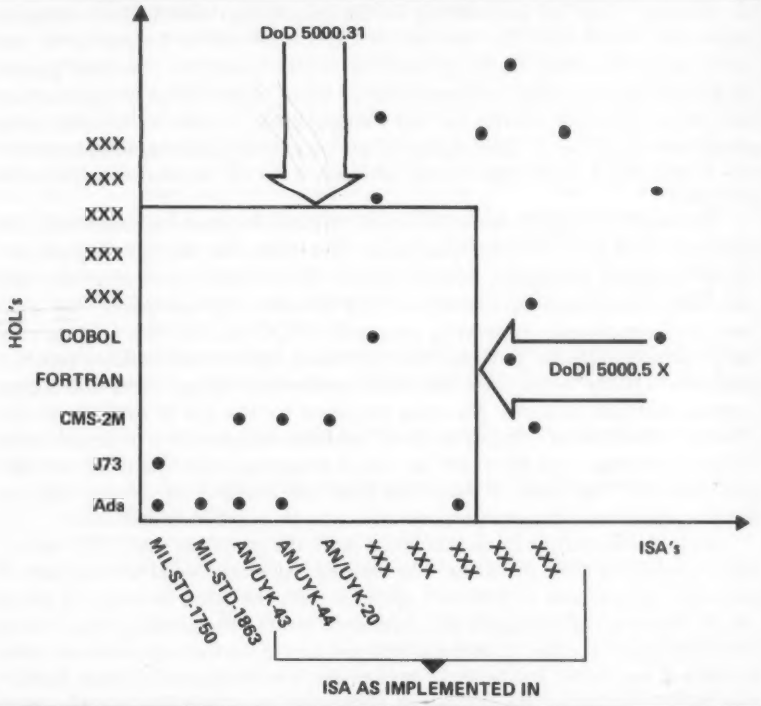
into the hardware area. However, the problems are not limited to software, but involve hardware as well and pose a dilemma as to whether more hardware may modify the need for software. As mentioned earlier, the cost of hardware is continually decreasing for an ever-increasing capability because of the large number of recurring kinds of applications in the commercial world. The computer population is exploding. The forecast for DOD is 260,000 by the year 1990, and could be astronomical in the commercial world when one considers games, pocket calculators, home computers and so forth. Construction of architecture, with its evolutionary nature, has not changed greatly, contrary to what some companies might claim. Examination of the primordial substrate of architecture will reveal that it is not significantly different from the original von Neumann concepts.[3]

The issue that arises is one of potential trade-offs between hardware and software that result from the increasing speed, decreasing cost, and growing population of hardware computers. Some claim that these factors concerning hardware may solve the software problems by killing software with hardware. That argument has been strongly resisted by responsible DOD officials, but only recently has a computer scientist published his supporting opinion that building complex hardware is prone to the same bugs and fundamental design errors that plague complex software systems. The main argument for the use of software in this dilemma is that once a complex design of hardware is committed, it is much more difficult to change, and therefore the cost of designing it the first time, the nonrecurring cost that then is amortized over the numbers of devices that are bought, will follow that same exponential curve that applies to software.

Figure 8 summarizes in an inferential form the problems and DOD thrusts toward resolving those problems. The dots are representative of intersections of language systems and architecture systems, with the latter in terms of either specific hardware or associated MIL standards. It should be noted that each dot in the matrix requires effort from the submerged part of the iceberg previously mentioned and represents a substantial investment. Department of Defense Instruction 5000.31 is forcing the number of high-order languages down to the seven preferred, while the new Instruction 5000.5X is pressing in on the number of instruction set architectures (ISAs). As a result, the numbers of intersections are being bound, whereas before these thrusts, there was a limitless number of possibilities. The issue is that if standardization can be achieved, and if systems can be described in a competitive fashion, a non-proprietary fashion, and if the technology can be improved, these "if" factors add up to drive down cost and

3. Named for John von Neumann (1903–1957), a Hungarian-born American mathematician who formulated game theory and was an early contributor to computer theory and computer design.

FIGURE 8
The M × N Problem



turn the exponential curve over during the 1980s to avoid the almost $40 billion investment of 1990.

The Standardization Issue

The case for standardization has been advanced fairly strongly above, yet the conventional wisdom says that standardization is the enemy of competition and technology. The word "conventional" is used, because it appears that the statement is no longer true—that if one were to standardize on a product that a large segment of the industry could possibly compete for on even footing, then costs do

decrease. Competition can be increased through that kind of standardization, and the injection of new technology can provide a competitive-edge parameter. For example, the Air Force is conducting a computer update on the F-111 models -D and -F because of the reliability problems associated with the computer. Initially, the computer was part of the original design of the MKII avoinics system, introduced in the 1960s. It is a relatively modest computer, with a current cost of $88,000 per copy. It was intended to procure an update from the original contractor on a sole-source basis, since the government did not own the design. Officials at higher levels in DOD, however, seeking to impose the principles of standardization and competition, requested a second look at the situation, and eventually the project was put out for competition.

As a result of that competition, several reasonably good bids were submitted, based on the Air Force's new MIL-STD-1750 architecture. This architecture had been implemented by a couple of contractors in a laboratory situation, although those contractors chose not to submit offers. Because the contractors experienced with the architecture chose not to bid, there were many who forecasted that the project would fail. However, there were at least a half dozen very competent bids from major companies, the winning one being awarded to a contractor who had no previous experience with MIL-STD-1750. The per-copy price of a new computer is $46,000, nearly half the cost of the original one, for some 500 systems. To be fair, the Air Force and industry had expected that the injection of new technology would have reduced cost in any event, their estimates being on the order of $57,000 per copy. The enforced decision to seek competition achieved savings beyond that the Air Force had envisioned.

The point that should be made concerning the F-111 computer is that while the savings are not such as to substantially alter the budget, still there was a strong indication that standardization on an architecture that is owned by the government, with several contractors who can enter competition at the same level of capitalization required, can provide significant savings. To carry that thought further, consider what savings could accrue if the Navy were to join for one of their future programs. The Navy is procuring some 10,000 copies of their standard airborne computer, which is basically the equivalent of the MIL-STD-1750 architecture, on a sole-source basis and at a cost of from $130,000 to $150,000.

The question arises concerning the freezing of design, and therefore the introduction of technology, at some point in the development stage. While this is an issue that derives from the standardization thrust, it is not one of great substance, because no matter how one arrives at the specific design that will be committed to hardware in quantity, there comes a time that the design *must* be frozen in order to invest in the production lines and the necessary support. The question remains,

does this just "happen," or is it the result of a conscious plan? We think the latter is better. It doesn't matter, so long as the design is reasonably modern. Also, implementation of designs and design changes on the basis of block installations should be considered. In the examples cited above, had an architecture that had been used in the past for some period of time been selected, then there might have been some limitations. However, the architectures that have been proposed are new and modern, one to five generations ahead of what is currently in the field.

Standardization does have to be approached on a rational basis. There are at least three levels at which standardization can be approached—in the hardware design, the instruction set architecture, or at the interface to the human element of the system, at the higher-order language and operating-systems level. It should be clear that the closer we bring standardization to the operating system, or hardware level, the less flexibility for change and the continued injection of new technology. Standardization can, of course, be accomplished on a block-change basis without changing the architecture.

To apply that approach further, a current program is being pursued to develop a microprocessor, a single-chip or one- or two-chip version of the MIL-STD-1750. Clearly, the technology employed will be a generation ahead of the F-111 changes cited above; so as the F-16 improvement program comes along, there will be generations in technology between them at the same architecture, hence enabling the capture of that software investment that was made earlier. Since there is a thrust toward this sort of standardization, i.e., a standardization of the interface between the hardware and its software, it is appropriate to define the term, "instruction set architecture" (ISA) as:

> The attributes of a computer as seen by a machine (assembly) language programmer, i.e., the conceptual structure and functional behavior as distinct from the organization of the data flow and controls, logic design, and physical implementation.

An ISA is *not*:

An implementation
A piece of hardware
A computer.

Don't be misled here; we are not advocating *use* of assembly language. This is merely the benchmark over which to set the transit through which we view the machine.

The thrust of proposed policy obviates the concern about locking into a particular technology at some point, because the policy addresses standardization of the interface rather than the hardware or a computer. The policy, contrary to the GAO advice to provide one language and one ISA to be used throughout DOD,

which is somewhat naive, is based on the success of limiting the languages to a small number and to apply that same approach to ISAs. Concomitantly, it is essential to provide a set of criteria for the selection of additions to the approved list of ISAs and for the deletion of items as they become technologically obsolete. If a direct-execution machine (one that does not require an intervening layer of activity between the software that a programmer produces and the hardware) becomes a reality in the future, then the policy should be such that it can accommodate the new machine.

The policy further encourages each service to propose ISAs and to develop implementing regulations. However, there must remain an oversight function at the OSD level. Although there are initiatives toward some decentralization of management within DOD, that oversight will be required. The rationale stems from the propensity of the various services, managers, laboratories, and so forth, toward the "not-invented-here" syndrome, and advances the thought that without oversight, the same job will be performed several times across DOD for no increased capability, yet the multiple costs, and it is that cost that needs managing. It should be pointed out that the policy does not propose directive oversight, but comparative oversight of activity. To reiterate, the reasons for ISA standardization are to:

—Reduce software resource expenditure over the system life cycle;
—Reduce development time;
—Reduce risk;
—Provide inter-system advantage.

Of these, the key concern is the reduction of risk and the reduction of development time and the effects those reductions can have not only on the psychology of the program manager's operations, but also on the actual costs. If development time can be reduced by concentrating available talent on the design process, then one can not help but save money, as has been demonstrated on at least two occasions.

The inter-system advantage accrues from "transportability." To illustrate that concept, consider another example from the F-111 program. During the life of the aircraft, there have been developed some 13 steering algorithms to tell the computer, or the pilot, how to guide the aircraft from Point A to Point B in space. The physics of that situation is not different, but the reason for iterating that many times is that developers numbered two through thirteen could not understand what the first developer did, or how he arrived at the design. The software was generally written in assembly language, and the originator tried numerous tricks to save resources and to do the things that engineers like to do. But as a result, the next series of developers were at a loss, requiring the additional iterations. Of course, some of the changes probably resulted from changes to the air-

craft itself, such as flight dynamics or power changes, or the methods of getting from A to B might have changed for various reasons—but not 13 times! Thus, the proposal is to capsulate some of those algorithms and use them across systems with the objective to saving cost.

To review, or list the potential advantages of standardizing ISAs:

—Existing support software used in muliple projects;
—Existing support facilities used in multiple projects;
—New software more transportable;
—Programmers need learn fewer ISAs;
—Software development need not await computer hardware selection;
—Run time software transferability enhanced;
—Technology and development focused on selected ISAs.

Thus, the proposed policy to be annunciated in DOD Instruction 5000.5X is as follows:

—Only DOD-approved ISAs will be used, *unless* it can be demonstrated that:
  -None is technically practical
  -None is life-cycle cost effective
—Each approved ISA will be assigned to a component to:
  -Assure stability
  -Provide configuration management
  -Certify implementation compliance
  -Disseminate information on ISA and software tools:
    -Government
    -Industry
  -Fully coordinate changes (DOD Directive 4120.3)
—List will be reviewed every 2 years minimum
—Waiver process will be published.

Coordination and Waivers

There are two issues of the policy that are important and worthy of further note. The first is that of coordination, which has not been particularly well done in the past. It has been said that "specifications are written in a broom closet and changed in a broom closet." Consider the effects on cost, for example, of non-coordination. The Army had decided to use the Air Force MIL-STD-1750 on a Missile Command project, without advising the Air Force of its intentions. The Air Force proceeded to change the architecture without consulting the Army through lack of knowledge that the Army was applying it. The resulting changes elicited as a result of that change will have to be implemented, a legitimate cause for contractors to recoup some costs from having "bought in" originally.

The second major issue is the one of publishing the waiver process. If a policy is going to be imposed that provides restrictions on the ability of a program manager to make decisions, then it is most appropriate to establish at the beginning the process in order to preclude *ad hoc* approaches, the growth of different sets of offices of responsibility, and other activities that stem from the bureaucratic process. In addition, it imposes upon the program manager a requirement to be specific as to the requested waiver—what is being waived and a good justification for the waiver. Some program managers, concerned with schedule and cost constraints on a major program, will desire blanket waivers, which will not be permitted, while some will blindly subject their programs to the policy when there might be strong justifications for a waiver.

The DOD-approved (DOD Instruction 5000.5X) ISAs include the following (note the change from the earlier use of the hardware nomenclature):

—MIL-STD-1862
—ELEX-P-351 (Navy standard 16-bit architecture)
—PD-PMS-408-1 (Navy standard 32-bit architecture)
—MIL-A-85232
—MIL-STD-1705A.

The AN/GYK-12 has been removed at the Army's request. It was their standard, but it employed technology from the 1950s and they do not want to employ it in new systems. In lieu of that, the Army will use MIL-STD-1862, which is the basis of the military computer family project beginning in 1983, and has issued strong direction to their commodity commands concerning its use. The Navy has yet to provide MIL-STD nomenclature, but is expected to do so in the near future. The MIL-A-85232 is the PROTEUS processor, and since it is a rather unique design and probably could not be rebuilt competitively, it is questionable whether it should survive on the final list.

The key in the ISA standardization issue, if companies are to compete on an equal basis, is that the design level in which we put "teeth" and control must be one for which DOD has appropriate rights to use and implement the ISA. It is inappropriate and unwise to commit to a proprietary product in advance of an objective and considered decision. That is not to say that proprietary products cannot be used, but that there must be strong justification that can be logically articulated as a basis for a waiver.

Objectives of Standardization

Most of the elements discussed above have been tailored from the general objective of standardization with DOD and will be criticized by those who consider the concept of standardization as inimical to the growth of capability. Quite to

the contrary, the objectives of the standardization program within the DOD are as follows:

—Improve the operational readiness of the military services;
—Conserve money, manpower, time, facilities, and natural resources;
—Optimize the variety of items, processes, and practices;
—Enhance interchangeability, reliability, and maintainability;
—Ensure that products are of requisite quality and satisfy minimum essential need;
—Ensure that specifications and standards are written to facilitate tailoring;
—Assure that specifications and standards imposed in acquisition programs are tailored to reflect only particular needs consistent with mission requirements.

Among the above objectives, it should be emphasized that a principal one is improved operational readiness, through improving logistic supportability and conserving money, manpower, time and so forth. Second, if things are used over and over again, with different views each time, then there is a smaller variety toward the optimization of which resources can be applied. A third important objective is the matter of providing quality products. One of the primary parameters of quality is the learning curve. The only viable method of climbing the learning curve is repetition, and standardization will generally enhance that. Finally, it is essential that specifications imposed are tailored to the individual requirements. The application should fit—not imposing a metrics capability on an inch-pound requirement, for instance.

Something should be said at this point about DOD initiatives in the computer-related standardization areas, which are listed as follows:

—CDNC Computer Aided Design Numerical Control
—IPSC Information Processing Standards for Computers
—SLHC Long-haul Communications
—TCTS Tactical Communications
—CMAN Configuration Management
—QCIC Quality Control
—ECR* Embedded Computer Resources

*A new proposed area

There is considerable standardization activity in those areas. The size of the standard data base in DOD for standardization and specifications contains some 45,000 individual documents, and they are the only ones that are legal to put on contract. Exceptions are interim standards or interim specifications or an *ad hoc* one that will apply to a single program. But if one is concerned with broad application over a span of programs, then the formal process of coordination and acceptance must be accomplished before they can be legal instruments on a contract. With the exception of the last category above, the areas apply to other than

embedded computer resources, and DOD's experience in those areas has been similar to its experience in systems acquisition, trying to utilize the General Services Administration (GSA), where their activities are not tailored to DOD's unique requirements. That is the reason for proposing the ECR area. It is merely an umbrella within the overall standardization and specification program, under which the small ECR region can be attacked. A forthcoming plan will identify those specifications and standards that are to be managed by a single office and all these areas are assigned to one of the military departments or to an OSD office for oversight. The plan will be circulated to the departmental standardization offices for a final coordination before it is issued, and there is certain to be some discussion over what should remain in the information processing area such as IPSC. There is considerable concern over these matters because of the legal aspects. A specification or standard must be assigned to a given property class before it can be legitimized, and if it is assigned to the IPSC area, it is automatically in Federal Stock Class 70 and therefore falls under the GSA guidelines. Such action is an automatic administrative action, and DOD might not realize the assignment until it is too late to correct. That is the principal reason for proposing a new area for ECR.

ECR Guidebook

For further information in the ECR area, one is referred to a Guidebook entitled: "Embedded Computer Resources and the DSARC Process." It is basically a question-type document, an outline, that is used at the Defense Systems Management College. Topics covered are as follows:

—Issues for DSARC Milestones I, II and III
—Definitions
—References
  -Regulations
  -Standards
  -Guidebooks
—A Feedback form

It provides the reader with some of the questions and some of the concerns that should be addressed during a DSARC or service-SARC review, depending on the system, and if those questions can be adequately answered by a program manager during preparation for the DSARC review, OSD Staff and other reviews, then he should have few problems. Although perhaps too much to hope for, if some of the elements outlined in the Guidebook were to be given attention during the formulation phase of a program, the level of capability with respect to computer resources should rise measurably. The purpose of the Guidebook was

to provide a greater emphasis for better resource management, although it also provides an educational kind of outline. It also may provide an outline for a syllabus for some of the training courses that are being contemplated.

Summary

In this paper I have tried to define ECR and deal with some of the problems associated with it, especially in terms of new and forthcoming DOD policy. Much of what I have addressed constitutes the unseen part of the iceberg, and this is the part we must continue to address if the future is to be assured.

But this is not the end of the story. There are some further refinements in process now, and subsequent articles will deal with these as the evolve. ||

37 Military Computers in Transition: Standards and Strategy

Louise Giovane Becker

The Department of Defense requirements for effective computer support systems have stimulated both innovation and controversy. Recent efforts to determine policies and standards, as well as certain acquisition strategies, have provoked a debate on the merits and disadvantages of the DOD approaches. Technical, commercial, and policy questions often have been intermeshed, thus causing some confusion and clouding of the issues. However, some major problems have emerged with regard to those computers that are integral parts of military equipment and weapon systems, or are used in "mission critical" operations.

Congressional interest in this subject is evident in legislation enacted in the 97th Congress and the continuing oversight and authorization reviews by committees in both Houses on Armed Services, Governmental Affairs, Government Operations, and Appropriations.

Nature and Scope of the Problem

A fundamental problem in determining a computer technology and standards strategy is to select a course that will ensure a potential enhancement or expansion of capability, but which does not disturb or negate ongoing activity. In short, an essential goal is to maintain a reasonable and flexible posture that meets present and future needs. This requirement is especially critical in defense and national security areas. The Department of Defense, as part of its continuing effort to maintain a battle-ready posture and expand its fighting capability, has supported efforts to standardize certain computer-related equipment (hardware) and the related computer programs and instructions (software). The DOD goal is to achieve optimal flexibility while obtaining the maximum level of computer technological advantage.

This paper provides a limited examination of some of the Department of Defense efforts and related armed services activities. Discussed here are representative activities that illustrate the DOD's efforts to standardize and manage mission-critical computer resources.

Louise Giovane Becker is a Specialist in Information Science and Technology for the Congressional Research Service of the Library of Congress. In this capacity she is responsible for providing information and consultant services to the Congress in the area of information technology policy. She was previously with Johns Hopkins University—Operations Research Office and Bell Telephone Laboratories. Mrs. Becker holds a B.S. degree in geology from Brooklyn College and has attended the University of Maryland and the University of California for advanced courses in science, telecommunications, and data network development.

Some Definitions for Understanding

The issues in this area are further complicated by unfamiliar terms and concepts. Much of the DOD standardization effort is directed to technologies referred to generally as "embedded computers."[1] While the most common definition of this term may include a broad spectrum of technologies from single chips to computer systems with multiple processors, it is also associated with an array of complex military operations and equipment. In order to illustrate the primary DOD standards program, this paper focuses primarily on computers that aid and support military operations, and which can be programmed without being subjected once again to the manufacturing process. This category of computer resources was delineated in the Department of Defense Authorization Act of 1982 (P.L. 97-86, Section 908), sometimes referred to as the Warner Amendment, which exempts certain classes of Defense computers from the procurement processes otherwise required by the Federal Property and Administrative Services Act of 1965 (P.L. 89-306).[2] This act includes computers and services for the conduct of the military missions of the Department of Defense related to:

—Intelligence activities;
—Cryptology activities related to national security;
—Command and control of military forces;
—Equipment integral to a weapon or weapons system;
—Equipment critical to direct fulfillment of military or intelligence missions, but
—Not routine administrative and business applications.

This category of computers required for the conduct of military missions is sometimes refered to as "mission-critical computer resources."

The term "embedded computers" has persisted in relation to "mission-critical computer resources," and this paper will use that term when it is consistent with such a reference or common usage. This paper also will use the term "mission-critical computer resources" when appropriate.

Major Issues and Concerns

Concerns regarding mission-critical computer resources have centered on the need for effective use of modern technology in support of combat readiness, and questions of the adequacy of the DOD strategy regarding the acquisition, development, and employment of embedded computers. For example, at a

1. This is a term that does not have a universally accepted definition, but is often described as "incorporated as an integral part of, dedicated to, or required for direct support of, or for the upgrading or modification of major or less-than major systems." Source: Department of Defense.
2. Generally referred to as the Brooks Act.

meeting earlier this year sponsored by the American Defense Preparedness Association Steering Committee on Military Computers and Software, questions were raised by industry representatives on the adequacy of DOD standardization programs.[3] Specifically, questions were raised on higher-order computer language standards, including one called "Ada"[4] which was developed by the Department of Defense, and standardization of the instruction set architecture.[5] While there was agreement that standards are useful, the level at which DOD proposed to implement standards was questioned by some at the meeting.

A related DOD goal is improving and expanding the present DOD embedded computer capability by eliminating obsolete[6] equipment; improving logistics support and maintenance functions; and assuring survivability and promoting interoperability.[7] The DOD standards program was designed to promote greater flexibility and competition while eliminating a "locked in" situation in which only one commercial vendor or one segment of the industry can adequately supply the needs of the military departments.

Another issue that has received attention regarding mission-critical computer resources is the recently mandated changes in procurement processes required by the Warner Amendment. Although the subject of computer procurement is beyond the scope of this paper, it cannot be entirely ignored in an examination of the present DOD standards initiatives. Congressional intent in both the Brooks Act and the Warner Amendment appears to be to stimulate and encourage effective acquisition of computer capability. Since experience with the new legislative requirement is limited as yet, it is not clear if DOD embedded computer resources capabilities will be improved by this new statutory requirement.

Some key questions that may require additional consideration include:

—Is the DOD acquisition strategy for embedded computers appropriate to meet present end future needs?

3. American Defense Preparedness Association MILCOM I. Military Computers and Software, January 27-28, 1982, Arlington, Virginia.

4. Ada is a trademark of the Department of Defense, named for Agusta Ada Lovelace, the first programmer.

5. A computer instruction set architecture can be described as the rules and procedures by which the hardware executes instructions or computer programs (software). (Source: conversations with Peter Fonash (DOD) and William Dietz (Carnegie Mellon University).) It also may be defined as the structure of a computer that a programmer needs to know to write time-independent machine language that will run correctly on the computer. (Source: the Department of Defense.) These definitions, although not universally accepted, are included here to provide a basic understanding of the concept.

6. Obsolescence in this context may refer not only to equipment that no longer meets requirements, but also to equipment that cannot be logistically supported.

7. Survivability means that, in battlefield situations, the equipment will withstand the environment. Interoperability means that systems will be standardized so that parts may be interchanged with other systems so that critical systems will endure and remain operational.

—Is DOD standardization program for embedded computers an appropriate strategy? At what level is standardization most effective?
—Has the concept of commercial vs. government-owned design of equipment and software for embedded computers been fully assessed?
—Have the technology and engineering investment issues regarding embedded computers been investigated adequately?

Management Policy Framework

The management of the Department of Defense embedded computer resources is a shared responsibility in which the individual services and the Under Secretary of Defense for Research and Engineering play a role, along with permanent and *ad hoc* joint organizational elements and groups. Generally, the individual services are responsible for the articulation of requirements and the program management of specific activities. The Under Secretary's office is responsible for coordination and management of department-wide policies for embedded computer resources and acquisition management. Within this office, the Deputy Under Secretary for Research and Advanced Technology provides the focus on embedded computer resources, including oversight of the standards for these resources. Groups, such as the Management Steering Committee for Embedded Computer Resources and the Ada Joint Program Office (discussed below), provide specific support and guidance. In addition, the Under Secretary for Policy and the Joint Chiefs of Staff provide coordination and validation in this arena.

Acquiring these resources also is subject to government-wide policies such as the Office of Management and Budget (OMB) Circular A-76, "Policies for Acquiring Commercial or Industrial Products," and OMB Circular A-109, "Major Systems Acquisitions."[8]

The defense management policy for embedded computers is articulated in DOD Directive 5000.29, "Management of Computer Resources in Major Defense Systems," which established a Management Steering Committee for Embedded Computer Resources (MSC-ECR) to provide appropriate oversight; DOD Instruction 5000.31, "Interim List of DOD Approved High Order Programming Languages (HOL)"; and in the proposed DOD Instruction 5000.5X, "Instruction Set Architecture (ISA) Standardization Policy for Embedded Computers."

8. A detailed review of federal policies is included in U.S. Library of Congress; Congressional Research Service. Federal Information Management Policy: Critical Directions; and CRS Report No. 81-101 SPR, authored by Louise Giovane Becker, Washington, 1981. p. 187.

Standardization Efforts

A major focus by the Department of Defense, as indicated by recent DOD initiatives, has been on standardization of embedded computer technology and software. These efforts are designed to permit greater savings in logistics support operations and to promote interoperability and survivability (as discussed above). While the focus on standards in this area stems mainly from logistics requirements, the desire to lower life-cycle management costs also has prompted consideration of a "standards approach." The services, therefore, have continued to promote a standardization program which includes both equipment and software.

Three distinct, but interrelated, initiatives that reflect the DOD strategy are as follows:

—The development of a military computer family (MCF) for the Department of the Army;

—Adoption of a higher order language, Ada;

—A proposed Department-wide instruction set architecture standard, Draft DOD Instruction 5000.5X.

Although the Department of the Army program is highlighted in this report, it should be noted that the Air Force and the Navy also support a standards program for their embedded computer resources. While the services' intent is similar, their approaches generally accommodate their specific mission.

Specific Initiatives

MILITARY COMPUTER FAMILY (MCF)

Beginning in the mid-1970s, the Department of the Army began a more intensive examination of its mission-critical computer requirements. As a first step, an attempt was made to obtain from the commercial sector an available instruction set architecture through a licensing agreement. After extensive examination, the Army concluded that a licensing arrangement was not advantageous. The adoption of a commercial instruction set was perceived by the Army and industry as placing unnecessary technical and administrative restrictions both on other participarting vendors and on the Army. The scope and nature of protections on the commercial instruction set architecture and the affiliated software also were viewed as a possible hindrance to the broad usage being contemplated by the Army. In addition, the Army concluded that the acceptance of a single firm's commercial instruction set might provide an unfair advantage to one company or segment of the industry and therefore restrict competition over a long period.

The Department of the Army contracted with Carnegie Mellon University, in 1979-80, to develop a computer instruction set architecture. This is referred to as

the Nebula instruction set architecture and designated as MIL-STD 182A. Nebula is acknowledged by some experts as an effective and advanced instruction set architecture. The Army and Air Force have worked together under a memorandum of agreement to develop and control the Nebula architecture.

The acceptance of the Carnegie Mellon University-developed ISA (Nebula) led the Army to consider using an innovative phased-acquisition approach in developing a computer family. Consequently, the military computer family program utilizes a multiphased competitive strategy. As a first step, 12 companies competed for the advanced development contact to produce a prototype model. Ultimately, four awards were made to five companies (Raytheon, General Electric/TRW, RCA, and IBM). It is contemplated that when this phase is completed, two of the five companies will be selected to develop a full-scale pre-production model. Ultimately, in a final competition for the production phase, one company will be selected. The production phase has a requirements contract that permits acquisition of units based on costs negotiated as part of the final competition. In addition, the MCF program strategy calls for technology infusion, that is, the continuing introduction of new technologies when they become available. The requirement for infusion of new technology is meant to ensure that the military computer family will have the latest in technology when the equipment is put into the field. Consequently, the Army is hopeful that it will improve survivability and that logistic management will be facilitated. The standards elements of the MCF program are designed to decrease costs for the initial units as well as to decrease costs over the life of the systems.

High Order Language—Ada

As part of its attempt at improving software standards, the Department of Defense has undertaken the development of a new programming language called Ada. Beginning in 1975, a DOD working group defined its higher-order language requirements and, after conducting a public review that found existing languages inadequate, initiated development of Ada.[9]

The management of the implementation of Ada is the responsibility of the Ada Joint Program Office, a permanent office under the Deputy Under Secretary of Defense (Research and Advanced Technology). This office coordinates all Ada-related activities within DOD and initiates automated support systems to facilitate Ada software development.[10]

As discussed above, the Department of Defense has issued two key sets of guidelines related to higher-order language standardization. Directive 5000.29

9. Department of Defense Ada Joint Program Office.
10. *Ibid.*

urges the use of one of a number of specified high-order languages unless cost savings through the whole life cycle can be achieved by using existing computer language techniques. Department of Defense Instruction 5000.31 specifies the approved languages which are approved and in an upcoming revision and Ada will become one of these languages. Ada represents the state-of-the-art and is to be controlled in both the development and implementation phases. As the DOD instruction is accepted, Ada will ultimately be the preferred language that will be used for most DOD embedded computer resources.

As a high-order language, Ada is not without problems or limitations. It is recognized generally that Ada, designed for a broad spectrum application areas, may not satisfy the needs of all applications. Since existing computer systems have software and computer programs in other languages, conversion is not always feasible or practical. Therefore, Ada is expected to be applied where practical, and waivers will be permitted as needed, but generally deviations will be discouraged. This standardization initiative may have transitional difficulties not only because of existing investments in software, but because Ada itself is in transition. Therefore, given that computer systems and the affiliated computer programs are in transition, all the advantages of this DOD standards effort program may not be realized for some time.

In an article published in 1981 in the IBM publication *Technical Directions*, it was noted that Ada had received an extensive review by the computer science and software engineering communities and its design is considered to be state-of-the art.[11] The article also noted that advantages in utilizing Ada have to do with software development support tools, that is, elements which support software development. Specifically, the article noted that "by standardizing a design language and by requiring a consistency of the design language with the language scheduled for predominant future use, the same tools used for programming may be used for design." Since these software elements are expensive, standardization would eliminate duplication and should result in savings.[12] The IBM Corporation is moving to implement a version of Ada and has concluded that the use of Ada will increase flexibility and aid in the greater utility of its software products.[13]

Ada's acceptance in the commercial sector also is indicated by the fact that the Ada language is in the final stages of consideration by the American National Standards Institute.

---

11. Douglas W. Waugh, "The Role of DOD's Ada in Software Standardization," *Technical Directions*, Vol. 7. Summer 1981, p. 43.
12. *Ibid.*
13. *Ibid.*

5000.5X INSTRUCTION SET ARCHITECURE (ISA)

Instruction set architecure has been described as those attributes of a computer as viewed by the machine language programmer. Further, it can be described, and has been by the Department of Defense, as the specification of the interface between software and hardware elements. As noted above, it may be viewed as the rules or procedures which allow software and hardware to interact. The ISA is in most, but not all, instances independent of the technology used.

The Department of Defense, as part of its overall embedded computer strategy, has encouraged the developmenat of a standard instruction set architecture. At the present time, a proposed Department of Defense Instruction No. 5000.5X (Draft) is under review. The instruction is designed to provide policy and guidance for the standardization of instruction set architectures for defense embedded computer systems and applications. The intent of such standardization is "to improve the overall effectiveness of DOD Computer resource utilization, including management of funds, manpower, time, and operational effectiveness."[14]

The DOD ISA policy has been discussed by the industry and reviewed by the Defense Science Board and the General Accounting Office (see below). Primarily, the debate on the proposed policy ISA has centered on the appropriateness of the level of standardization being proposed in DOD Instruction 5000.5X. At the heart of the matter is speculation as to whether this level of standardization will exclude the military departments from obtaining innovative technologies and limit them from future developments.

Observations on DOD Standards Strategy

The DOD embedded computer standards programs have been commented on by industry, the General Accounting Office, and the Defense Science Board (DSB) Task Force on Embedded Computer Resources Acquisition and Management. While in some instances certain reports of these groups are not available at this writing, some public comments have been made of selected findings contained in these reports.

As early as the spring of 1981, the Under Secretary of Defense for Research and Engineering, Dr. Richard D. DeLauer, was asked questions by members of Congress and representatives from industry regarding the strategy concerning the utilization of commercial computer technologies and, more specifically, the proposed DOD Instruction 5000.5X, "Instruction Set Architecture Standard."

---

14. U.S. Department of Defense, Department of Defense Instruction Number 5000.5X, Draft (not dated).

Letters to Dr. DeLauer expressed concerns about the Defense standards program and its potential to ensure that up-to-date technologies would be employed in embedded computers. Questions were raised on the need for 5000.5X, the ability of the policy to meet future defense needs, the limitations on DOD purchase of commercial products, the effective utilization of new technologies, and the likelihood of restricting competition. In responding to the individual letters, Dr. DeLauer expressed confidence in the DOD strategy and pledged to seek coordination with the private sector through its principal industry associations. Subsequently, Dr. DeLauer requested that the Defense Science Board convene a task force to address these issues in detail.

DEFENSE SCIENCE BOARD TASK FORCE

As previously noted, the DOD convened the Defense Science Board Task Force on Embedded Computer Resources in 1981 to review the proposed DOD Instruction 5000.5X and associated problems. Consequently, the DSB Task Force in a business notice in *Commerce Business Daily*, October 23, 1981, solicited comments on four major areas. Industry was requested to respond to the following questions:

—Are the management policies enunciated in DOD Directive 5000.29, "Management of Computer Resources in Major Defense Systems," DOD Instruction 5000.31, "Interim List of DOD Approved High Order Programming Languages (HOL)," and proposed DOD Instruction Set 5000.5X, "Instruction Set Architectures (ISA) Standardization for Embedded Computers," appropriate to the present?

—Are the key embedded computer programs of the Department—Ada, Nebula, NECS, MIL STD 1750—properly constituted and supported?

—Are the organizational issues of consequence and, if so, what recommendations can be advanced for improvement?

—What is the effect of the legislative and regulatory environment upon the ability of the Department of Defense to make adequate use of digital technology?

At the American Defense Preparedness MIL-COM meeting on January 27 and 28, 1982, discussed above, the Defense Science Board Task Force on Embedded Computer Resources Acquisition and Management announced its support of the proposed DOD Instruction 5000.5X. Although the final report of the Task Force has not been officially released at this writing, the support for standardization at the ISA level seemed to be accepted with some qualifications. At the same meeting, the General Accounting Office announced in a GAO letter report that it had concluded that the DOD ISA policy would not meet the needs of Defense in the future.

GENERAL ACCOUNTING OFFICE LETTER REPORT

The GAO letter report on DOD Instruction 5000.5X, prepared at the request of Chairman Jack Brooks of the House Committee on Government Operations, evaluates Defense plans for implementation.[15] The GAO report concluded that contemplated changes in commercial architectures and support hardware and software "should compel [DOD] . . . to further evaluate the level of standardization to be achieved" before further commitment. The letter report views ". . . the need for architecture standardization as a function, in part, of the availability of Ada as the standard computer program language." The report goes on to note that "because Ada is being developed to be a machine-transportable [equipment independent, that is, the language will operate on many makes of computers] language with a relatively low life cycle maintenance cost, the need for standard architecture may be diminished when it is available."

The GAO letter report recommended that the Secretary of Defense not implement the DOD Instruction 5000.5X, and it also went on to recommend that the "services reevaluate their ongoing efforts and demonstrate why they are more cost effective than standardizing on a high-order language such as Ada and relying instead on the computer industry to provide the stimulus for computer architectural innovations." Subsequently, Chairman Brooks wrote to the Secretary of Defense urging that Defense reconsider the proposed instruction.[16]

Conclusions

The increased dependency of the military departments on embedded computer resources as reflected in the level of automation in military equipment has raised a wide spectrum of concerns, including concerns about the appropriateness and adequacy of certain standards, initiatives, and strategies.

Critics of the DOD approach have been concerned that some of the initiatives being taken or proposed will tend to restrict defense capabilities. Specifically, concerns have been voiced over proposed DOD Instruction 5000.5X, an instruction set architecture policy which would place controls and limit the variety of equipment and associated software that can be employed.

The Defense Science Board Task Force report, which is scheduled to be released shortly, at this writing, purportedly will support the instruction set architecture policy. At this time it is not clear what other aspects of the DOD strategy will be supported by the Task Force.

15. U.S. Comptroller General of the United States, Letter Report. Subject: "Standard Instruction Set Architectures for Embedded Computers," (MASAD-82-16), January 27, 1982, Washington, General Accounting Office, 1982, p. 3.

16. Letter to the Honorable Caspar W. Weinberger, Secretary, Department of Defense, from Chairman Jack Brooks of the House Committee on Government Operations, January 27, 1982.

Congress has continued to review the embedded computer resources standards problem. Hearings were held by the House Committee on government operations, chaired by Representative Jack Brooks. Subsequently, the Defense Authorization Act language required that the Department of Defense review further the standardization of instructional set architectures. In addition, both the Senate and House Defense Appropriations Committees have the issue under consideration.

While these reviews are significant, the Department of Defense faces a limitation of time. Given the dynamic nature of the military computer requirements and the quickly developing technology, the DOD decisions cannot be postponed indefinitely. Defense, faced with rapid changes in computer technology and software, may be caught in a dangerous dilemma in which any action, in retrospect, may be deemed inappropriate. Some remedy may lie either in enhancing the existing DOD management and coordination functions or by establishing a permanent advisory group and appropriate staff to provide management oversight.

At the heart of the matter is whether an appropriate decision can ever be made when the technology, the basis for the standard, is evolving and transitory. In brief, can standards be adequately designed and implemented to provide an appropriate level of uniformity without stifling innovation and limiting access to desirable new technologies? There is very little doubt that additional experience with the multiphased competitive contracting and technology infusion process is needed to aid DOD in making appropriate decisions. Unfortunately, there may not always be time to reach consensus or arrive at a perfect set of decisions. In order to obtain some coherent DOD strategy, it may be necessary to provide a consistent approach that will meet immediate and middle-range requirements, an approach that will have some flexibility so that provisions for changes and technological infusion may become more advantageous for the Department of Defense. ||

Suggestions for DOD Management of Computer Software

48

AIA/EIA White Paper

In the earliest applications of digital computers to military systems, the computers provided accurate computational assistance to otherwise analog systems. The associated software served essentially as the instruction book necessary for the computer to perform its functions, and the cost of this software was a minor portion of the total system development cost.

Digital computer technology developed rapidly following the emergence of semiconductor integrated circuits, and the computers in military systems were called upon to perform an increasing variety of both computational and control functions. Included in this variety was the provision of multiple modes of system operation, either in automatic sequence or on optional command. And as the scope of the computer functions expanded, so did the magnitude, cost, and relative visibility of the software. The software still was generally regarded as the instruction book for the computer, however, and typically was procured as a data item on the system development contract.

As the world of information-handling electronics became more and more universally digital, embedded computer software became an increasingly substantial fraction, and frequently a dominant fraction, of military system development effort and cost. Partly because of this relative magnitude and partly because of the not fully matured evolution of the digital processing and software technologies and disciplines, software has frequently been a pacing item in the development of military systems. Furthermore, because software appears esoteric and somewhat mysterious to many managers, problems associated with it tend to produce a greater level of anxiety than problems of comparable severity in the more tangible and familiar hardware.

A more subtle complication in the management of software-dependent systems, however, stems from a fundamental reversal in the roles of the software and the electronic hardware that has taken place without being universally recognized. As substantial digital processing resources began to appear within essentially all electronic subsystems, it naturally developed that (1) these digital processing resources almost always incorporated the central control functions of their respective subsystems, and (2) the primary channel of inter-subsystem communications was between these digital processing resources, typically under software control via a common data bus. Thus the combined distributed software, which previously would have been perceived as the set of instructions required

---

Editor's Note: This AIA/EIA White Paper was prepared by the Technical Management Committee of the Aerospace Industries Association (AIA) in early 1979. The paper then proceeded through a series of AIA/EIA reviews and revisions, culminating in the current version, which was released in April of 1982. The AIA *Ad Hoc* Computer Resources Technology Group and the G-33 Computer Resources Data and Configuration Management Committee of the Electronic Industries Association (EIA) were responsible for the development and release of this paper.

for the various digital computing elements of the system to function, could now be recognized as essentially *being* the system, containing within itself the complete description of what the total system is supposed to do and how. In other words, the software embedded in electronic systems has moved from the role of servant to that of master, and the electronic hardware elements (not limited to the embedded computers) are now best understood as the tools necessary to enable the software instructions to be carried out.

Therefore, modern military system management, both customer and contractor, is confronted with a difficult but not often clearly understood paradox: Being a crucial and dominant deliverable element of the system, the software clearly warrants at least the same degree of development discipline, quality assurance, and configuration management as the hardware. On the other hand, because the software encompasses the detailed functioning of all facets of the entire system, it cannot be characterized early in the program by subsystem-like functional requirements and interfaces and then developed independently as can the classical subsystems. In other words, while the deliverable software obviously requires disciplined development, its content cannot effectively be subjected to earlier or greater constraint than the overall system engineering process itself.

The Environment of Rapidly Advancing Technology

The rapid growth and dynamic nature of data processing development are well known. Software development methodologies must continue to evolve and be improved to keep pace with the increased capabilities of hardware and the increased expectations of the user. The major factors affecting the software development environment are advances in hardware capability and cost, increased complexity of application requirements, and progress in software development technology and tools.

The ever-increasing speed and storage capabilities available with general-purpose computers have made possible the implementation of very complex real-time systems such as air traffic control and battlefield command and control. Advances in distributed processing and the availability of microprocessors have brought special-purpose processors with architectures optimized for specific functional capabilities such as array processing. Increased user needs include expanded real-time requirements and an increased user dependence on the results of the processing.

Software engineers have recognized the environmental influences on requirements and the lack of both discipline and effective yardsticks in software development methodology. New tools and techniques are being developed and an environment that permits and encourages application of these new ideas is necessary if methodology is to keep pace with increased requirements.

MITRE/APL Identification of Key Problems and DOD and Industry Response

MITRE AND APL STUDIES

In 1975 the Johns Hopkins University Applied Physics Laboratory (APL) and the MITRE Corporation conducted studies for the Department of Defense concerning DOD weapon systems software management. The findings of 10 previous major DOD-sponsored studies relating to computer software were available to the MITRE and APL study teams. The MITRE Corporation concluded that the major contributing factor to weapon system computer software problems was a lack of discipline and engineering rigor applied consistently to the software acquisition activities. More specific findings identified by MITRE included the following:

—Frequent contributors to software cost and schedule growth include: (a) poorly formulated initial software requirements; (b) changing requirements and requirements growth during the development phase; (c) false starts and need to educate involved organizations before useful output can be obtained; (d) inefficient use (proliferation) of already existing resources; (e) inefficient testing and verification tools and methods; and (f) improper use (poor tailoring) of standards and guidance documents in specific procurements.

—There is a general need for better definitions of software terms, measures of software qualities, and the methods of measuring them.

—Software technology improvements particularly aimed at developing a software engineering discipline are being made by industry, academia, and the services but require application to real military systems (in addition to laboratory or experimental systems) for evaluation and confirmation.[1]

The Applied Physics Laboratory study resulted in 17 recommendations, each of which was intended to address a specific problem. Among those recommendations were the following:

—Specify that major computer software involved in weapon systems development be designated "configuration items" and be deliverable during full-scale engineering development.

—Use top-down design, i.e., specify the use of modular software architecture and an orderly, phased design approach for developing major computer programs that defines the higher levels of the program and then progresses to design and test successively lower levels.

---

1. MITRE Corporation, "DOD Weapon Systems Software Acquisition and Management Study," Vol. I, Findings and Recommendations, May 1975.

—Require the contractor to apply a highly disciplined set of engineering practices to the detailed design and programming phase of development. This must include a clear and disciplined set of standards covering program structure, size, control, interface, formal conventions on data base management, and demonstration that the standards are enforced in practice.
—Establish a common set of requirements and criteria to be applied in the acquisition and support of weapon system computer resources by all services. Recommended AFR 800-14, Vol. II, as a start.
—Prepare a series of handbooks or guides covering important aspects of software acquisition to help program managers to define, review, and evaluate requirements, procedures, proposals and designs during pre-contract and contract management.[2]

GUIDEBOOKS, DODD 5000.29, OTHER DIRECTIVES

During the 4 years following completion of the MITRE and APL studies, some 25 volumes of guidebooks, plus a number of directive documents dealing with software development and acquisition, were issued by DOD and the military services. Also in existence at the beginning of this period were MIL-S-52779 dealing with software quality assurance,[3] an Army manual on data processing systems,[4] and three Navy documents on data system and computer program documentation requirements.[5]

The guidebooks included 24 dealing with all facets of software development and acquisition management published by or under the direction of the Air Force Electronic Systems[6] and Aeronautical Systems[7] Divisions, plus one published by DOD on embedded computer resources.[8] The principal directives concerning embedded computer resources issued during this period were DOD Directive

2. Johns Hopkins University, Applied Physics Laboratory, "DOD Weapons Systems Software Management Study," June 1975.
3. MIL-S-52779 (AD), "Software Quality Assurance Program Requirements," 5 April 1974.
4. USACSC Manual 18-100, C-2, "Tactical Automatic Data Processing Systems Development, Maintenance, and Documentation Standards," 9 October 1974.
5. WS-8506, "Requirements for Digital Computer Program Documentation," 1 November 1971; SECNAVINST 3560.1, "Tactical Digital Systems Documentation Standards," 8 August 1974; and SECNAVINST 5233.1A, C-1, "Department of the Navy Automated Data System Documentation Standards," 30 August 1974.
6. Air Force Electronic Systems Division, "Software Acquisition Management Guidebooks" (17 volumes). October 1975 to March 1978.
7. Logicon, Inc., "Management Guide to Avionics Software Acquisition" (4 volumes), June 1976, and TRW, "Airborne Systems Software Acquisition Engineering Guidebook" (3 volumes), November 1977.
8. Department of Defense, "Embedded Computer Resources and the DSARC Process—A Guidebook," 1977 Edition.

5000.29[9] and DOD Instruction 5000.31,[10] Air Force Regulation 800-14,[11] and the Navy Mil-Standard 1679.[12] Also issued were a DOD standard on data system documentation,[13] and a Rome Air Development Center general specification on software development.[14] Revision and updating of these directives and regulations is an ongoing task. In addition, DOD Instruction 5000.5X on instruction set architectures is in the process of coordination.

INDUSTRY RESPONSE

Partially in response to this mass of directives and guidebooks, but more fundamentally in response simply to the basic need, the various software development contractors have been developing more and more completely the disciplines and procedures required to develop effective software. Although the government guidebooks have been of some usefulness in this regard, their primary utility has been to provide a source of education and guidance for the government program managers and other service personnel involved in software acquisition, and to provide a basis for acquisition, as opposed to *development*, discipline, and standardization. The combination of technical sophistication and continuingly advancing technology makes it almost axiomatic that the technical capabilities and practices of the bulk of the software development contractors will lead, rather than depend upon, the government technical documents.

JOINT LOGISTICS COMMANDERS WORKSHOP

In April 1979, the Software Management Subgroup of the Joint Policy Coordinating Group on Computer Resources Management (JPCG-CRM) sponsored the Joint Logistic Commanders Software Workshop, involving technical personnel from the services, other federal agencies, and industry. The stated purpose of this workshop was to examine the services' software acquisition guidelines, management procedures, and standardization efforts to determine if there was a basis for coordination and adoption of joint-service standards.

---

9. DOD Directive 5000.29, "Management of Computer Resources in Major Defense Systems," 26 April 1976.
10. DOD Instruction 5000.31, "Interim List of DOD Approved High Order Programming Languages (HOL)," 24 November 1976.
11. AFR 800-14, Vol. I, "Management of Computer Resources in Systems," 12 September 1975; Vol. II, "Acquisition and Support Procedures for Computer Resources in Systems," 26 September 1975.
12. MIL-STD-1679 (Navy), "Weapon System Software Development," 1 December 1978.
13. DOD Standard 7936.1-S, "Automated Data Systems Documentation Standards," 13 September 1977.
14. RADC, "General Specification for Computer Software Developmment," CP 9787796100D, 31 March 1978.

The principal findings of the workshop were as follows:
—Differing policies exist among the services, producing differences in emphasis and nomenclature with varying interpretations and degrees of implementation.
—Within DOD, there are a number of diverse regulations and standards covering the various aspects of software acquisition. Partial and inadequate standards coupled with a lack of standardization of data item descriptions (DIDs) led to delivered software products that are often unsatisfactory.
—Difficulties in the implementation of software quality assurance (SQA) arise from the lack of well-defined consistent requirements, differences in SQA approaches by the services, and unavailability of experienced personnel.
—Unlike hardware, successful development of software cannot be based simply on passing a definitive series of tests at the end of development; however, there is no standard set of software acceptance criteria within DOD.[15]

Key recommendations in the final report of the workshop were to:
—Develop a general policy framework for the joint services to address the entire software life cycle.
—Develop a single, unified set of acquisition and development standards for joint-service application.

As a result of these recommendations the JPCG-CRM took upon itself a plan of action. This plan of action included five tasks:
—Generate a plan for software acquisition policy.
—Develop a single, unified set of acquisition standards for tri-service application.
—Develop a comprehensive set of DIDs, subsets of which would be applicable to any DOD software contract.
—Generate a data item description for a contractor's software quality assurance plan as a tri-service DID.
—Define and develop software acceptance policy, procedures, and criteria for the acquisition of software in defense systems.

Each task has been initiated. A draft software acquisition policy is currently being reviewed prior to publication. Draft standards for tasks two and three exist, and publication for review was expected in the first half of 1982.

In June of 1981, the same group sponsored a second workshop that also included technical personnel from the services, other federal agencies, and industry. The purpose of this second workshop was to continue the discussion and to review additional documentation and software management areas of standardization within DOD. Five panels were established for this purpose:

15. "Final Report of the Joint Logistics Commanders Software Workshop," 1 October 1979.

—Data item description study review
—Hardware/software/firmware configuration item selection criteria
—Standardization and accreditation of computer architectures
—Estimating software costs
—Software reusability

A draft report of the proceedings of this workshop was issued in November of 1981.

Software Management Challenges in the Face of Accelerating Technology Change

The preceding paragraphs have outlined the technological and management history of embedded computer software development for DOD applications. In the light of this history, five major challenges to future DOD and industry management of military embedded software development have been identified:

THE NEED FOR SOFTWARE MANAGEMENT DISCIPLINE WITHOUT INHIBITING EFFECTIVE DESIGN

Both the Johns Hopkins and the MITRE studies identified the central need to convert software development from an individualized art form into a disciplined engineering activity. This need was a direct consequence of the continually increasing quantity and criticality of defense software, coupled with a general need for the delivered software to be functionally expandable and modularly modifiable or reusable.

Despite the need for manageable and verifiable software design, quality, and configuration disciplines, however, the technology of digital hardware and systems, as well as of software itself, has continued to develop dramatically. The critical management challenge, therefore, is to implement the requisite degree of discipline without inhibiting the exercise of innovative design competence or application of the new and emerging technologies.

For quite fundamental reasons, significant differences tend to exist between DOD and industry approaches to this challenge. These differences stem from (a) the inherently differing roles of customer and supplier, (b) the typically different levels of first-hand familiarity with software development between the two, and (c) the dual role of embedded software itself.

Through continual engagement in the actual development of software and software technology, each of the principal software suppliers has developed detailed, disciplined procedures for software management. These procedures are, of course, not identical from firm to firm, and furthermore are constantly developing as the technologies of computing hardware and software itself continue to develop. Therefore, customer imposition of an arbitrary set of manage-

ment or design procedures, for a particular program or in general, may appear disruptive or even counterproductive to the contractor.

On the other hand, the military services and their acquisition commands are faced with the necessity of dealing with many suppliers, large and small, and have had painful experiences with cost, schedule, and system performance problems ascribable to inadequate discipline in the development of embedded software. In addition, the service program managers frequently do not themselves have a first-hand working knowledge of software. Therefore, there is a strong motivation on the part of DOD, and each of its subordinate elements, to seek a commonality of approach that can be required of all suppliers, and that can facilitate the education and training of its own personnel.

A further complication is the inherently dual role of embedded software in digital systems: (a) the obvious role as a deliverable element of the system, requiring at least the same degree of quality assurance and configuration management as the hardware elements of the system; and (b) the not always clearly perceived role as the primary documentation of the detailed functional design of the system, which documentation should not be overly constrained until the detailed design is, in fact, complete. The differing configuration management implications of these two roles tend to be not nearly as forcefully evident to the acquiring customer as they are to the development contractor.

The dichotomies cited above can be expected to widen, of course, as technological change continues to accelerate. It is perhaps evident that the key to resolution of this inherent problem lies in distinguishing carefully between acquisition disciplines and design disciplines.

THE NEED FOR FLEXIBLE STANDARDS FOR SOFTWARE OF DIFFERING SCOPE AND APPLICATIONS

Another management challenge stems from the practical and economic need for different approaches to the development of software of differing scope and applications. The differing degrees of management structure required depend on the magnitude of complexity of the software being developed. There can also be significant differences in the standards that are appropriate depending on the mission application. For example, the standards applied to software for a low-cost tactical missile should permit a high degree of design flexibility to minimize the recurring hardware cost, whereas in a high-level command system, it is cost-effective to use hardware less efficiently to minimize the software cost. A related example is the difference between custom-applications software and multiple-usage support software. The latter may, on occasion, be available commercially with validation provided by extensive prior usage.

THE NEED TO ACCOMMODATE CONTINUING COMPUTING TECHNOLOGY ADVANCES

A central challenge to the development of software management standards and discipline is the continuing stream of advances in both the hardware and software facets of computing technology. On the hardware side are both the continual dramatic decrease in cost per function and the continuing emergence of new elements and architectures such as the microprocessor, array processors, non-volatile memories, and alterable firmware. On the software side is a similar continuing emergence of new techniques, logical structures, and tools such as structured design, high-order languages, packet switching, distributed processing, fault-tolerant architectures, and an ever-growing array of software tools for the computer aiding of software development.

THE NEED TO DEFINE REQUIREMENTS FOR MODIFICATION OF DELIVERED SOFTWARE

In the development of military hardware, it normally is assumed that the customer will be called upon to maintain it in operable condition, but will not be called upon to redesign it subsequent to delivery. The customer therefore typically requires delivery of maintenance handbooks and pertinent test equipment but not, for example, the computer-aided design programs used in the development of the delivered hardware.

There is no such widely accepted "normal pattern" relative to embedded software, however. Because ones and zeros do not fail or wear out, software has no need for maintenance in the repair sense. On the other hand, one of the key features of software is its ability to be modified (i.e., redesigned) relatively easily to accommodate changes in the mission requirements or the desired modes of system operation. If the customer wishes to be able to accomplish such post-delivery modifications (frequently, though confusingly, called "maintenance") himself, then the software must be designed from the beginning to facilitate such customer modifiability. And in particular, the customer may indeed need to possess or acquire the pertinent software development tools used in generating the originally delivered software.

It follows that the customer's post-delivery-modification requirements may have a significant impact on both the appropriate application software design requirements and the requirements for documentation and delivery of support software.

THE APPROPRIATE LEVEL, CONTENT, AND INTERRELATIONSHIPS OF DEFENSE SOFTWARE SPECIFICATIONS AND STANDARDS

In view of the management challenges cited above, it is evident that effective structuring of the necessary defense software specifications and standards calls for careful integration of diverse and sometimes conflicting factors. In particular,

it would be valuable to minimize unnecessary differences in the stated requirements of the various services and their subordinate commands, as recommended by the 1979 Joint Logistics Commanders Software Workshop[16] and to facilitate the joint participation of industry and government in developing the documentation of such requirements.

The following sections discuss this and the other identified management challenges in greater detail, and offer some recommended DOD policy actions in connection therewith.

DISCUSSION OF MANAGEMENT CHALLENGES

The Need for Software Management Discipline Without Inhibiting Effective Design

It is the opinion of the AIA and the EIA Government Division that the government is overconstraining contractor development approaches for embedded computer software. These constraints result in increased software cost and risk when contractors are forced to use practices different from their normal approaches or are prevented from using new innovative management techniques appropriate to the rapidly advancing data processor and software development technology.

THE DIVERGENCE BETWEEN DOD AND INDUSTRY APPROACHES

In 1976 the DOD issued Directive 5000.29, "Management of Computer Resources in Major Defense Systems." This document required that the services "prepare and maintain appropriate guidance documents covering requirements definition, development, acquisition, operation, and support issues attendant to computer software in defense systems." Since publication of the directive, more than 35 guidance and requirements documents have been published and more are being prepared. These government documents cover every phase of software development from design through coding, testing, and maintenance.

During the same time period the majority of major DOD software development contractors generated and installed systems of practices and procedures assuring that a disciplined approach is followed in all their software development. These practices and procedures, like the documents issued by the government, cover all phases of software design. While these contractor practices differ throughout industry and from those developed by the government, the objectives of all are the same—to produce software of acceptable quality at the lowest cost.

16. *Ibid.*

The development of contractor practices has continued in an attempt to further reduce cost as new tools and management approaches have been proposed. These attempts include, for example, the "chief programmer team" and the "software factory." Major emphasis has been placed on extending the number of development options and tools, tailored to the job at hand, available to a software development manager. The government, on the other hand, has tried to *limit* the number of available approaches. This is the most fundamental difference in the industrial and governmental approaches to software development problems.

MISAPPLICATION OF PROGRAM MANAGEMENT GUIDANCE DOCUMENTS

As government regulations and guidebooks have been developed, contractors have received increasing pressure from government program offices to adopt the practices detailed in these documents, rather than those developed by the contractor. In some cases the documents have been included as contract compliance documents. In our experience, driving a contractor to a different development approach from his established and installed practices increases program risk and cost significantly. Also, with the number and wide variation in documents and their application among the various services and their elements, there is a growing danger that contractors will be faced with using a different management approach for each program directed by a different government agency or even program office.

DISTINGUISHING BETWEEN ACQUISITION POLICY AND DESIGN METHODOLOGY

It is the opinion of the AIA and the EIA Government Division that if the government policies were tailored to *acquisition* rather than *development* details, this conflict would be minimized and acquisition costs and risk would be reduced. For example, direction of the form and substance to be included in software requirements documents is appropriate and will minimize products that do not meet user requirements. The government in this case is detailing how the government-to-contractor interface will be controlled. Dictating the structure of the software by specifying exact interfaces between computer program configuration items (CPCIs) early in the program, however, is not appropriate for the government, because it is important for the contractor to be free to adjust these interfaces, internally to his design, as the design matures. The dictation of what higher-order language (HOL) to use (which may limit hardware choices) and instruction sets (which will limit hardware choices) can be questioned on similar grounds. These latter limitations stem primarily from the desire of the government to be able to modify (called "maintain" in most government documents) software after delivery. This will be discussed in more detail later.

The technology advances that have dramatically driven down the cost of data processing hardware are also driving new concepts in software development. In the 1970s, software development support systems depended on time sharing of a large and expensive central processor among many development personnel. This large investment led to development concepts that stressed extensive review of design prior to cost and test, to minimize each programmer's use of a precious resource—CPU time. The day is now upon us when for a very low cost a 2-3 million-instructions-per-second (MIPS) machine will be available to each programmer. Minimizing the use of this machine is no longer a proper approach to development policy.

However, many of the government's software development policies today are based on this approach; for example, requiring that no coding be accomplished prior to the critical design review (CDR). This process normally involves review of large amounts of design documentation (not products) by contractor and government personnel based on predictions of its performance. This approach is intended to prevent waste of resources (man-hours and data processor time) in developing code that does not meet requirements. Present wisdom, however, is that it would be better to prototype at least portions of the software (comparable to breadboarding hardware) and present test results at CDR, because the cost of data processor time to accomplish this is very low, and the cost of development man-hours is probably more than balanced by the reduction of documentation and the increased definitiveness of the review. This type of new development concept is a direct outgrowth of new hardware developments. Government policies should not prevent development approaches that are evolved from advances in technology.

This same hardware trend has caused the dominant system architecture to shift from large centralized processors to mini- and microprocessors networked together to perform required weapon system functions. These networks demand distributed data bases, and therefore large transfers of data by the processors. The throughput of the networks is significantly affected by these data transfers and by the network protocol necessary to retain system sanity. The creation of the network operating system must be accomplished early in the program; e.g., at about preliminary design review (PDR) time, in order to provide a development environment for the remainder of the application programs. Again, while the contractor's practices are rapidly being tailored for this evolving environment, government policies are, in many cases, still based on a philosophy of development more appropriate to a large centralized data processor.

These examples are not intended to be critical of government initiatives for improvement of the software acquisition process, which are certainly ap-

propriate. What these examples have tried to point out is that freezing development practices and procedures is not the appropriate tactic to satisfy the government's goal; that, rather, it is in the government's interest to encourage continual evolution of software development practices based on emerging new management concepts and hardware and software tools.

MAINTENANCE OF DISCIPLINE WITHOUT UNNECESSARY DESIGN CONSTRAINT

On the other hand, it is vital that there be no reduction in the discipline of software development that has been gained over the past decade. Any contractor without established practices for software development management, acceptable to the government, should have to develop such practices or adopt available practices during the competition for any major software development program. We suggest that a policy of audit of such practices, rather than dictation of those practices, is in the best interest of both the government and industry.

In summary, while contractors driven by competitive pressure have been examining additional development approaches and management methodologies to attempt to reduce development complexities and cost, the government has been attempting to systematically *reduce* the number of choices available. We are not convinced that either government or industry has the wisdom to settle on a small number of development options in a field as dynamic as embedded data processing or software for weapon systems.

The Need for Flexible Standards for Software of Differing Scope and Applications

The rigidity of some standards and the lack of the ability to tailor others pose some difficult challenges, particularly for contractors working on a variety of embedded computer software projects.

THE ACCOMMODATION OF VARYING COMPLEXITY AND APPLICATIONS

The size of the software task is universally recognized as a contributor to complexity because of the increased number of interactions of both people and program segments. Yet, existing government standards do not recognize that differences in program size should influence the decision to apply standards. In fact, the requirement to impose specifications on subcontractors ("flowing down") further amplifies this problem. Thus, a prime contractor faces the necessity of deciding between a fully responsive flowing down of requirements or a more risky but cost-effective tailoring of the requirements.

In addition to size, complexity may be introduced in other ways. For example, a decision to put a segment of the software into read-only memory adds com-

plication in defining the deliverable form of the program and identifying configuration items. Similarly, employment of a processor that uses a combination of new applications software and standard commercial packages complicates the full application of a development standard such as MIL-STD-1679 to the software development process.

It also is appropriate in evaluating the applications of standards, to distinguish between the more routine, well-defined types of software such as standard scientific functions, tape and disk utility programs, etc., and the less routine, more complex types such as operating systems, executive and control programs, a data base manager for a distributed processing system, or a device controller for non-standard peripheral equipment (displays, encryption/deciphering equipment). Yet, there are virtually no provisions within the standards to accommodate these differences. As a result, management is faced with the challenge of either accepting the standard in its entirety with consequent major inefficiencies or, after a thorough review process, compiling a list of exceptions for each software item.

CONTROLS ON SUPPORT AND TEST SOFTWARE

The software developer is similarly challenged when asked to apply standards originally intended to prevent problems in operational software, to support and test software. These standards, although very important to the software development process, are not in the same category as operational software when viewed from the perspective of the weapon system manager. The failure of a mission as a result of a software deficiency is the direct result of an error in the operational program. There is a low probability that such a deficiency will have been the result of an error in the support software that was not detected by diagnostics and testing. There is a much higher probability that the error was introduced during the design/development process for the operational program itself. To apply the standards for operational software development (including interface, development, and product specifications) to support software does not seem appropriate or cost-effective in many instances.

The handling of off-the-shelf software is a specific challenge. Compiler development often precedes the awarding of a contract so that contractor-developed support may already exist. However, documentation in accordance with government specifications is usually not done by a contractor using company funds. Unless the support software is properly considered during the procurement process, there could be unnecessary problems during the delivery phase when the government representative is trying to determine if the contractual requirements (including software development specifications) have been met. Additional complications occur with proprietary or licensed software.

Test equipment plays a key role in the overall weapon system procurement, and the trend is toward increased use of programmable test equipment containing microcomputers/microprocessors and associated software. There is no question of the importance of this test equipment to the overall program; however, to put the same kinds of documentation, development, and quality controls on this embedded software as on the embedded operational software seems inappropriate. Such software must be controlled and documented in accord with appropriate design and quality assurance procedures, however.

DESIGNATION AND HANDLING OF COMPUTER PROGRAM CONFIGURATION ITEMS

A final area in which flexibility is important for effective software management relates to the handling of computer program configuration items. Here, also, existing standards lack the flexibility required to establish a range of controls for computer programs. The choice is to either (1) identify the program as a computer program configuration item (CPCI) and impose the related documentation and control specifications such as MIL-STD-483 and 490, or (2) not identify the program as a CPCI and obtain contractor documentation only. Certainly there should be some more appropriate middle-of-the-road position. For example, consideration should be given to the use of a form of configuration management that includes baseline controls, specifications, identification, and change control after integration testing, but excludes plan preparation, prescribed-format status accounting, and government CPCI and CPIN numbering systems.

Premature, arbitrary partitioning of a system/subsystem into a number of "chunks" or subprograms with their own documentation, reviews, tests, etc., by the government procuring agency too often preordains computer program design and is counterproductive to the computer software development. Partitioning into CPCIs should be recognized as a technical task to be performed by the developing organization as an integral part of the design process. The CPCIs should be kept to a manageable minimum, and decisions regarding the number should be technically driven and made by the developer with the concurrence of the program manager.

The Need to Accommodate Continuing Advances in Computing Technology

Examples of the direct impact of computer technology advances in software development methodology have already been discussed in this paper. Of yet broader significance is the impact of technology advances in the embedded computer resources of which the software is a part, and in the systems in which they are embedded.

THE IMPACT OF HARDWARE AND SYSTEMS TECHNOLOGY ADVANCES ON SOFTWARE DEVELOPMENT METHODOLOGY

The rapid growth and dynamic nature of data processing technology are well known. Software development methodologies must continue to evolve and be improved to keep pace with the increased capabilities of hardware and the increased expectations of the users. As the hardware has become more powerful and more imaginative, requirements have been defined, and software complexity has increased faster than the software development technology and tools to meet those requirements. An environment encouraging the development of more effective software technology is required if this technology is not going to continue to limit the rate of systems progress.

The hardware side of the data-processing industry is charging ahead with refinements that continue to provide ever-more computing power and storage capability. For example, advances in distributed processing and the availability of microprocessors have brought us special-purpose processors with architectures optimized for specialized functional capabilities, such as array processing. Higher-order language availability for such special purpose processors is limited. Where execution time is critical, for example, in signal processing applications, the processor implementation may be highly parallel with a horizontal microcode programming language. Available HOLs fall short of the need, offering little capability to address the parallel processing problems in distributed processing. Hardware advances are projected to continue at a high rate, placing additional demands on the need to develop and apply new software technology and tools.

Software technology also will be driven by advances in basic systems technology needed to meet new classes of mission requirements. For example, embedded computers are at the heart of defensive and weapon delivery subsystems with rapid response real-time requirements which effectively exclude a man in the loop. As the integrity of the processing becomes more critical, for example, in processes that control weapons release, the need for error tolerance increases. Systems must be tolerant not only of hardware errors and previously undetected software errors, but also of operator or input induced errors, whether the erroneous input is accidental or results from a willful attempt to hamper proper system operation. These factors will affect both system complexity and basic system architecture, as input data rates continue to increase and distributed processing brings about more and more computer-to-computer communication with no practical opportunity for timely monitoring by a human. It is evident that corresponding advances in software technology, tools, and techniques will be required.

THE IMPEDIMENT OF OVERLY DETAILED SPECIFICATIONS AND STANDARDS

As software development methodology has evolved, the standards intended to ensure development of a quality product have required update and change. Much of this change has been required because standards have addressed in too much detail how a procedure should be carried out instead of emphasizing what the objectives and attributes of the end products should be. Some specific examples are as follows:

—Design languages have permitted program designs to be recorded in a format that permits computer-aided maintenance. The design language can also be carried along in program source listings as program commentary. This helps ensure that the design continues to be updated as source changes are implemented in the program maintenance phases. Standards regarding flow chart requirements and conventions are not needed and can discourage the use of design languages.

—In distributed processing systems with special purpose microprocessors, the procedure and standards governing software development on the various processors may need to differ. Whereas an HOL may be highly desirable for the general processing elements, the use of the same or any HOL for some of the microprocessors may not be feasible. The requirement that a contractor demonstrate a life-cycle cost benefit is at best difficult and may be impossible.

—Problem Statement Language/Problem Statement Analyzer (PSL/PSA) is a program specification language that permits precise program specification and computer-aided analysis of the quality of the specification. However, a PSL specification deviates from the generally required English text format for software specifications.

—The need for maintenance tools and documentation also varies depending on the application. When the program storage is implemented in read-only memory (ROM) that can be changed only at the factory, the maintenance requirements are obviously very different from those normally encountered.

Independent of application, software development will continue to require a precise statement of requirements, traceability of requirements through design and implementation, comprehensive validation and effective documentation, and tools for maintenance. The procedures used to attain a quality product will vary with application and processor type and must be permitted to evolve to keep pace with the expanding challenges.

The Need to Define Requirements for Modification of Delivered Software

It is the opinion of the AIA and the EIA Government Division that a major opportunity for improved control over the cost and performance of systems involving embedded computer software is possible if the unique problems relating

to post-delivery modification of the software are properly recognized and treated.

THE NEED FOR AND NATURE OF POST-DELIVERY SOFTWARE MODIFICATIONS

One of the primary advantages of software-controlled systems is the relative ease with which the functioning of the system can be changed by "simply" modifying the software. Such changes may be needed to implement new mission requirements, to incorporate or counter new technologies, or perhaps just to make the system meet the original requirements. However, current practices for modifying delivered software systems frequently result in excess costs, failure to realize performance potential, systems out of action for unreasonable lengths of time, etc. The root cause of these problems is a massive failure, on the part of customers and contractors alike to consider adequately or provide appropriately for the capabilities and resources necessary to carry out such post-delivery modifications.

The capability to make post-delivery software changes has clearly been the intent of the customer in many cases and of the contractor in some. But well-thought-out provisions to carry out these changes have been the exception. Frequently, either no explicit provisions are made, or else the myth of considering software changes as "maintenance" is used as a means of masquerading their true nature.

Every post-delivery change to software is actually a change to the design of the delivered package. It may be anything from a trivial change to a highly significant one, and it may result from a change in requirements or the need to make the design meet the original requirements; but in any case, it is a design change and must be treated as such. Furthermore, there may be absolutely no correlation between the size or "ease" of a change to a software package and its impact upon the performance of the system in which it is embedded.

The all-to-frequent description of post-delivery change activity as "maintenance" can be more damaging than making no explicit change provisions at all, because it may deceive the customer into thinking the problems are under control and thereby dissipate both customer and contractor resources in working the wrong problems. In fact, ones and zeros do not deteriorate or fail as mechanical or electronic components may, so software has no need whatsoever for maintenance in the usual repair sense. (Software can be changed through the action of electrical fields or particles on the media containing it, but this is clearly a hardware problem.) The ease with which software can be changed may be part of the reason for the perpetuation of the "software maintenance" myth, there being a tendency to equate easy-to-do tasks, regardless of their consequences, with "fixing" something.

POLICIES AND PROCEDURES NEEDED

When even relatively modest changes to a hardware system are envisioned, it generally is recognized that significant design and system engineering effort will be required and that there are substantial performance, cost, time, and interface factors to be considered. While the nature of software makes the mechanics of physically changing it easy, the other fundamentals are not changed. Changes still must be designed, interfaces considered, documentation created, tools made available, people trained, etc. The determination of what, how, when, and by whom changes to delivered software are to be made are also vital matters. In short, although the change may be only an invisible rearrangement of some electronic states in a memory, the considerations are exactly the same as for any other design changes.

Development of software that is intended to be modified only up to the time of delivery may be substantially less complex and less costly than development of software intended to be modifiable after delivery or deployment. The data base structure used, architecture of development hardware, documentation, training, use of proprietary software/firmware/hardware in development of the deliverable software, etc., can be affected by decisions about whether and how future modifications are to be provided for.

There is a need for the customer and contractor to cooperatively determine very early in the acquisition process how much future flexibility in the system design is to be provided. Because substantial trade-offs of design philosophy, data base design, hardware architecture, etc., are involved, this matter must be tackled, if not resolved, during concept validation.

SPECIFIC CONSIDERATIONS IN PROVIDING FOR POST-DELIVERY MODIFICATION

Provision for future changes is fundamentally to allow for partial or total redesign of the originally delivered software. Consideration must therefore be given to identifying, preserving, acquiring rights to, preparing documentation for, and maintaining configuration control over those development tools, aids, programs, etc., that must be available to carry out the redesign activity.

Selecting the necessary items is not a trivial process, because even the well-disciplined development of software remains, to a significant degree, a creative process. Software engineers, like all other engineers, use a multitude of specialized, temporary and intermediate aids which may facilitate the task of creation for themselves, for their particular group, or in certain circumstances. These same aids may be of little use to different individuals or in other environments. To attempt to insist that no such aids (tools) be used is totally impractical in the absence of complete automation of the software development process. On the other hand, indiscriminate requirements for the delivery of all tools used in the development of the original design would add enormous and unnecessary cost.

Also, redevelopment or modification by different people at a different time or in a different environment frequently may be done more effectively using different tools than those used originally. Such alternatives must be considered.

The actual process of changing the system after delivery also must be considered during acquisition planning. Questions of who should or can make such changes; how multiple copies of such changes are to be redistributed and controlled; whether and how multiple versions of changed software are to be handled; whether or how new utilities are to be used; scope of changes to be made by customer field personnel or others, as opposed to those requiring involvement of the originator; responsibility of the originator for system performance with non-originator-changed software components, etc., must be addressed.

THE MOST URGENT NEED

What is not urgently needed is early and wide appreciation of the true nature of the software product and its post-delivery modifiction. Both customer and contractor must realize that a product whose design is intended to be, and in many cases will be, changed after delivery is a very different type of product from that common in the past. Recognition of this difference should then permit establishment of the general and specific acquisition and development policies and procedures that are necessary if the customer is to get the maximum value that is possible and practical.

The Appropriate Level, Content, and Interrelationships of Defense Software Specifications and Standards

The current situation relative to the level, content, and relationship of the various military software specifications and standards is, unfortunately, confusing and frustrating. The situation is further confused by the existence of tutorial type documents (guidebooks) that are often designated as compliance documents, despite the absence of any thought-out determination as to how, if at all, they should fit into the total specifications and standards hierarchy. The management challenge presented by this plethora of specifications, standards, and guidebooks is illustrated in the following examples.

One challenge is provided by MIL-S-52779A, "Software Quality Assurance Program Requirements," dated 1 August 1979, which has recently been adopted as a joint-service standard. This standard attempts to legislate an adequate software development by imposing historically hardware-manufacturing-oriented tools, techniques, and methods on the development process. Although the need for effective software quality assurance is unquestioned, it also is quite clear that it cannot be achieved by the same conceptual approach as that applicable to production hardware.

A second challenge is the excessive how-to detail imposed by some specifications and standards. For example, MIL-STD-1679 (Navy), "Tactical Software Development," imposes a specific working organization, dictates the conduct of programmers and their supervisors, and requires excessive documentation. The document dictates how work is to be done in much greater detail than is required to assure compliance with actual customer needs.

A third challenge stems from application, as contractual documents, of the "Software Acquisition Management Guidebooks." Although the apparent intent of these documents was to provide guidance to the procuring agency in acquiring software, the documents are frequently being directly applied contractually. DODD 4120.21 expressly forbids the use of non-product documents as contractual requirements unless they are listed in DODISS (DOD Index of Specifications and Standards) or AMSDL (Acquisition Management Systems and Data Requirements Control List). It is our view, however, that enforcement of this directive has been lax.

A fourth challenge is presented by the requirement of DOD Directive 5000.29 that all computer programs be treated as configuration items. Misinterpretation of this has resulted in the premature involvement of a host of individual service-oriented requirements documents, many of which have been previously identified as cost-driver specifications. Although the totality of deliverable software for a military system clearly should be treated as one or more computer program configuration items, the premature designation of a particular computer program as a CPCI can result in:

—Increased paperwork involved in preparing, processing, and status reporting engineering changes;

—Burden of maintaining interface control documentation;

—Increased development time and costs (e.g., conducting audits, reporting status, preparing plans and specifications).

RECOMMENDED POLICY ACTIONS

Relative to the Need for Discipline Without Inhibiting Effective Design

Acquisition policies should not dictate design methodology. Acquisition policies should concentrate on providing the best possible set of requirements at the earliest possible time. There is still a lack of early trade-offs between the hardware and software solutions to a portion of a defense system. Almost universally, contractors are trying to generate the most complete consistent and testable set of requirements early in the software development cycle. It is clear that this is the single most effective way to minimize software development cost.

Guidebooks should not be applied to contracts as compliance documents. Guidebooks are intended to help the government program managers and not to serve as "how to" books for contractors. Major software contractors have made large investments in developing policies, in training their people in following those policies, and in software development tools consistent with those policies. Audit those policies for compliance with government requirements, rather than arbitrarily imposing others.

Detailed pro forma management plans should not be required from contractors with established acceptable practices and procedures. Require copies of published practices for quality assurance, configuration management, cost accounting, and the like, as is done in the cost and schedule control system criteria (CSCSC). Depend on audits of contractor's existing/established development policies.

Relative to the Need for Flexible Standards

It is recommended that DOD, in establishing new standards (or revising existing standards, specifications and guidebooks), consider the following specific recommendations:

Specifications and standards should recognize differences in the use, size, scope, and complexity of software, and be organized to facilitate tailoring the document accordingly. Different levels of software (e.g., microcode, assembly code, high-level language code and design language) and different types of software (e.g., command and control or signal processing) have different development and documentation requirements. These differences must be recognized in DOD specifications and standards by defining requirements that can be appropriately tailored to specific procurements.

Existing specifications need to be reformatted to facilitate tailoring. The guidance for tailoring should be included as appendices or in the handbooks. The tailoring should be done through appropriate references in the statement of work or other contractual documentation.

Instruction sets and architectures should be allowed to be extended and modified where required for reasons of efficiency, particularly for systems such as ECM or missile guidance systems where large hardware production runs are anticipated.

Software configuration items (CPCIs) should be specified only after careful consideration of the system requirements. The premature designation of CPCIs should be avoided because of the additional (but unproductive) configuration reporting required. Formal configuration control can be required at the systems level, but contractor part number control will often suffice for software sub-

system designations and control. Identifying CPCIs selectively is essential to holding costs at a reasonable level. Preliminary design review is an appropriate time for formalizing the designation of CPCIs.

Supporting documentation and standard development procedures should not be required for support software unless a real need exists. Adequate user's documentation is generally available in contractor's format. Maintenance documentation should not be required unless the support software itself is intended for maintenance by someone other than the developer. Government development specifications should not generally be required for off-the-shelf, contractor-developed support software. Exceptions to this would occur when the support software is a contracted deliverable item such as an Ada compiler.

Relative to the Need to Accommodate Continuing Technology Advances

Specifications and standards should define only what is needed, and should not specify the methodology to be used by the contractor in greater detail than required to assure compliance with those needs.

Relative to Requirement for Modification of Delivered Software

DOD software acquisition policies and procedures must be revised through joint industry/government action to:

—Make clear that modification of software after design completion is conceptually different from the repair of failed hardware, i.e., that post-design modification of software is in most cases redesign and not "maintenance" in the hardware sense;

—Make clear that adequate provision for post-delivery modification requires special consideration during acquisition and development;

—Require identification during concept validation of the extent of and approach to post-delivery modification to be required, and of the consequent need for delivered software support components;

—Cause those specifications and standards affecting software to reflect differences among systems and support components to be used with systems intended for: (a) no post-delivery modification, (b) post-delivery modification, and (c) post-deployment modification.

Relative to the Appropriate Level, Content, and Interrelationships of Defense Software Specifications and Standards

It is recommended that DOD establish a new standardization area for embedded computer resources. A plan for standardization could outline objectives, priorities, milestones, and specific tasks. The DOD objective could be a minimum number of fully coordinated tri-service standards, in line with the recommendations of the 1979 Joint Logistics Commanders Software Workshop. An initial effort could be the development of an acceptable standards and specifications tree, which would be fully understood and accepted by both industry and government. The following are specific recommendations and support rationale that would, in our opinion, lead to a strong program.

Issue a tri-service standards and specifications tree for software. Standards and specifications for documentation and acceptance are absolutely necessary. They could contain software-peculiar content only. The tree could identify who has the prime preparation and maintenance responsibility at the levels of the DOD, tri-service, and service-peculiar. It would also identify the interrelationships of the various directives, standards, handbooks, guideline documents, *et. al.;* their contractual status; and their use for internal DOD purposes or external contractor purposes. It would be an effective planning and control tool to reduce proliferation, redundancy, and omissions. The tree could be controlled by the appropriate office within the Office of the Under Secretary of Defense for Research and Engineering (OUSDRE). A model for industry/government cooperation in defining an appropriate tree is that cooperation in addressing the subject of engineering drawings.

Develop criteria for issuance of software specifications and standards. Sufficient knowledge, experience, and directions exist to determine when a DOD or tri-service level document should be prepared, coordinated and issued, consistent with the Defense Standardization Manual DOD 4120.3-M. However, individual services, or portions thereof, are in need of guidance. The "when" and "what" of individual service standards and specifications initiatives deserve thought and dissemination. Typical criteria could be to issue such specifications and standards when:

—The identified DOD and tri-service level documents are inadequate (or missing);

—A recurring problem is clearly understood and a generally accepted solution is known;

—The solution has a successful history;

—The benefits would clearly offset the costs, i.e., apply a disciplined cost benefits analysis.

Involve a moratorium on the issuance of new single-service specifications and standards until recommendations 1 and 2 are in place. This would provide the time for the limited resources in industry and the government to step back and examine that which has been done and develop a more logical future plan.

Establish a joint government/industry board to define annually future specification and standardization tasks. The Aerospace Industries Association and Electronic Industries Association Government Division strongly support a cooperative approach in the subject of standards and specifications. The impact on daily operations is so profound that an extensive dialogue by informed persons would be advantageous. An annual review of the tree and definition of next-year tasks would be likely to result in the availability of senior industry personnel for participation in such a program. ||

73 Today's Risks in Software Development — Can They Be Significantly Reduced?

Major General Jerry Max Bunyard, USA
James Mike Coward

During the past 15 years, there has been a significant growth in the development of complex, real-time embedded computer systems, both large and small. With this growth, the government has incurred many development problems that have resulted in poor quality of the software products. Methodologies and tools have been generated with the goal of identifying problems, resolving them early, and producing reliable, maintainable products, but software overruns still occur, schedules still slip, and software products still fall short of their goals.

According to Dr. Barry Boehm, speaking at the Software Summit Series in Los Angeles in May 1980, the cost trends for both software development and software maintenance are rising substantially and are not projected to improve (Figure 1).

Emphasis continues to be given to cost for development, with little attention being given to the overall life-cycle cost. This leads to a further rise in maintenance costs due to latent errors in the software and inflexible software design, as well as insufficient documentation and support software for maintenance. Support packages, e.g., simulations and test tools, are often treated as throwaways rather than major deliverable items that facilitate both development and maintenance. More emphasis must be placed on the required discipline and rigor early in development to ensure quality, including maintainability of the software throughout the system life cycle.

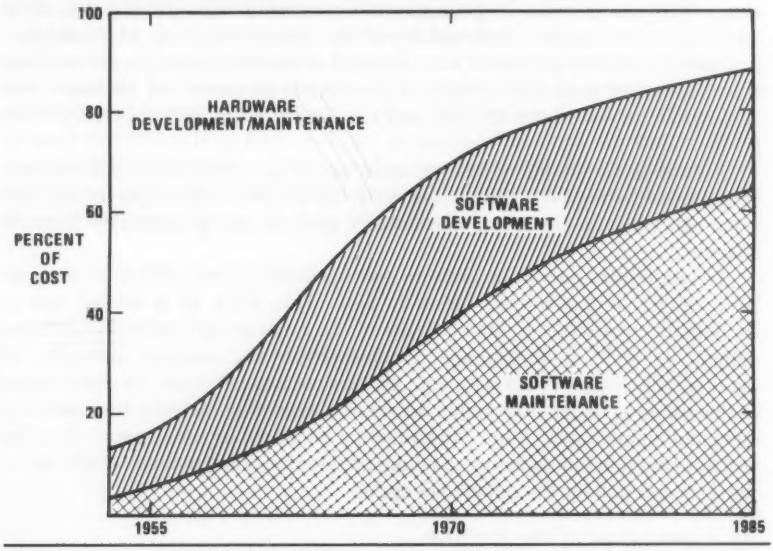
An Examination of the Problem

This article takes a look at today's risks in software development, with the objectives of focusing on lessons learned and recommending alternatives that can significantly reduce these risks. What are the primary problem areas today that lead to schedule slippages, cost overruns, or a software product that falls short of its desired goals? Some of these problem areas are original requirements that are incomplete and/or validated; software design that is not traceable to the requirements and diverges during development; software code that is not maintainable due to poor enforcement of standards; documentation of the system that

Major General Jerry Max Bunyard, USA, is Project Manager, Patriot Missile System, at Redstone Arsenal, Ala. He was previously Deputy Director of Tactical Air and Land Warfare Systems, Office of the Director, Defense Test and Evaluation, within the Office of the Under Secretary of Defense (Research and Engineering). General Bunyard holds a B.S. degree from Oklahoma State University and an M.S. degree from George Washington University.

James Mike Coward is Manager, Data Systems Department, at Teledyne Brown Engineering, Huntsville, Ala., where previously he was Project Manager for the Patriot Software V&V Project. Mr. Coward holds a B.S. degree from Lamar University and an M.S. degree from Texas A&M University.

FIGURE 1
Hardware-Software Cost Trends

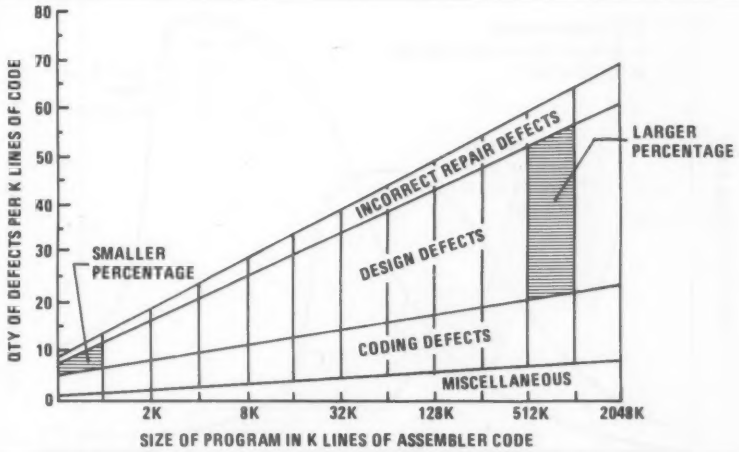


does not reflect as-built code; software that is insufficiently tested; and timing and storage budgets that are exceeded.

In software development, specific software errors are frequently classified into one of three categories: requirements errors, design errors, or coding errors. This classification is a natural result of the three major development phases that lead to fabrication of the software components. Figure 2, presented at the AIIE 1977 Software Conference, illustrates that as larger software products are considered, the number of design defects begins to exceed the number of coding defects, until design defects predominate the total errors. This implies that larger numbers of more complex defects are being introduced early in the development cycle of larger software products.

The software development cycle is often presented as a sequential set of well-defined phases, each with specific products and reviews, which provide the necessary structure to facilitate management and control by the materiel developer and project manager. Software development activities on many proj-

FIGURE 2
Overview of Potential Life-Cycle Defects in Programming

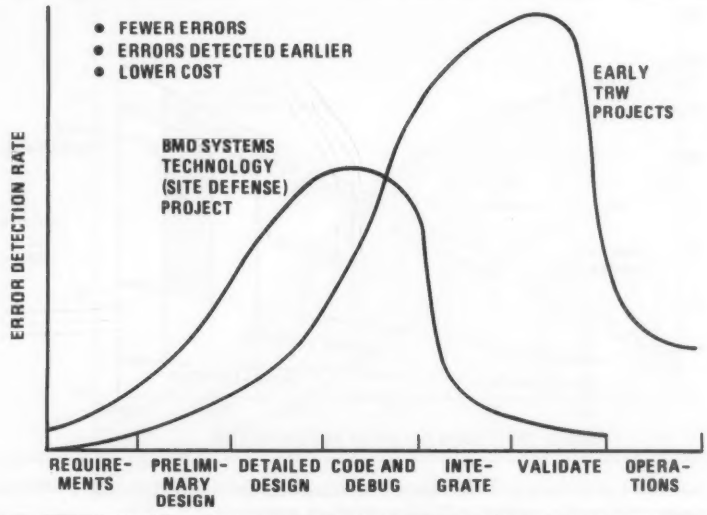


ects, however, continue to be burdened with management and technical problems and issues that work against a quality product end-item.

In contrast, it is important to emphasize that techniques and methodologies have been defined and are being applied to some projects. They are contributing significantly to the delivery and support of quality products. As an example, in the early 1970s, a structured, disciplined approach strengthened by employment of an independent evaluation contractor contributed significantly to reducing software risks on the Ballistic Missile Defense Systems Technology Project. As illustrated in Figure 3, presented at the 1976 AIAA Software Management Conference, the total number of errors was reduced and the error detection rate for this project was substantially shifted, resulting in early problem identification and resolution.

When large numbers of problems surface late in the development program during system integration and test, the development effort is impacted by major requirement and design issues. In Figure 4, presented by Dr. Jacques Gansler, then Deputy Assistant Secretary of Defense for Materiel Acquisition, at the 1977 AIAA Software Conference, the increased cost of correcting a requirement or design error as one progresses further into the development is illustrated. These data, used with the detection rate curves presented in Figure 3, illustrate very

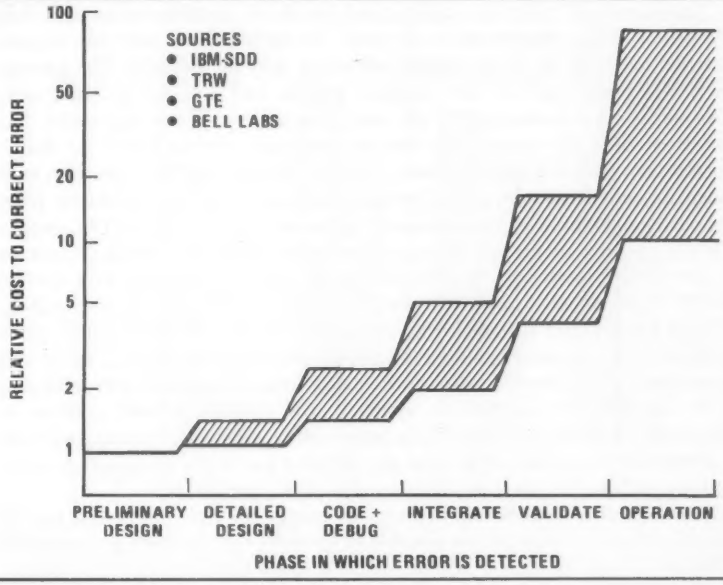
FIGURE 3
Catching Software Errors Early: Project Results



forcefully the risks associated with software costs: i.e., large numbers of errors found late in the development cycle. When errors are found late, requirements have to be revalidated, designs redesigned, software and systems retested, and documentation rewritten.

The previous paragraphs describe a major contributor to unexpected software costs. Planners don't project the costs to re-do what was not done right the first time. Since the code appears to be something that can be modified with relative ease, problem identification and resolution are often postponed until after the coding phase is completed. There is a need to recognize that there is more to fixing and maintaining software systems than making a few coding changes. Significant software risks are incurred when requirements and design flaws are not given sufficient attention early in the development cycle.

FIGURE 4
Catching Software Errors Late: The Cost



Software Quality—It Begins with Procurement

THE PROCUREMENT ENVIRONMENT AND ITS IMPLICATIONS

First, let us look further at the environment in which the software development contractor works. The pressures of competition for development contracts and program funding constraints work against quality development practices as early as the proposal.

Every contractor in the world would like to propose the most thorough design job and best documentation conceivable. In fact, in costing exercises before proposal submission, such an approach might even be typical for a first cut. Competition for development contracts is intense, however, and this places extreme pressure on writers of cost proposals. This competitive pressure may also be aggravated by extremely difficult funding constraints imposed by inadequate appropriations from Congress. The result is usually that the dollars bid for software

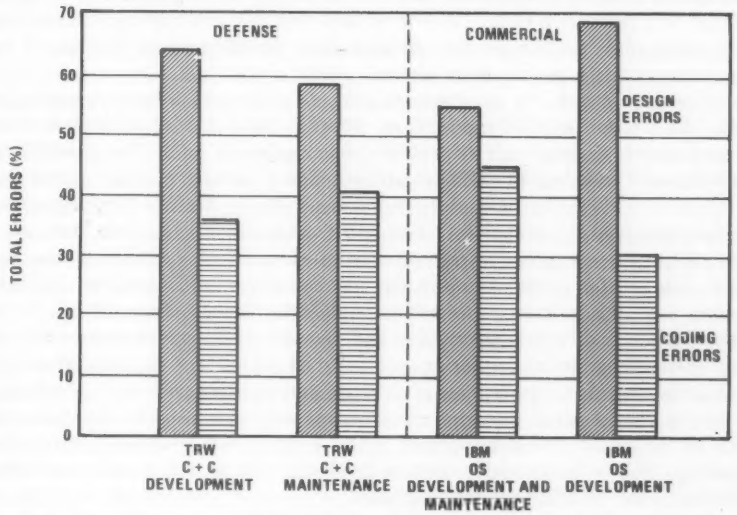
development will be cut before proposal submission and during negotiations, especially at the time of best and final offer. In making these price cuts, the contractor typically assumes that the military standards applied to the development are "loose enough" that the quality and depth of documentation and data presented at design reviews can be cut back, the requirements work can actually be carried over into the design phase, and so on through the cycle. The assumption is also made that the test program will go well and can be sized to fit whatever dollars and schedule are left when development reaches that point. This assumption basically means that the test program will be based on dollars available, not technical thoroughness, and that the tests will be completed independent of product quality. This typically will leave software problems to be fixed during operations and maintenance, after product acceptance. The contractor is also counting on his people to put in whatever hours are necessary to get the job done and is counting on contract modifications associated with new or modified requirements to help compensate for the initially insufficient funding.

When the contract is signed, the contractor is faced with still another problem: staffing. There may be only a few people left from the proposal effort who are familiar with the project. Also, a sufficient number of required personnel may not be immediately available to assign to the project to meet staffing requirements. Thus the contractor starts understaffed, facing both hiring and training problems. This most likely was not planned for in the proposal, so work schedules will start to slip from the first day of the project.

Then begins the series of requirements and design reviews. The contractor will generally present the best picture possible of the work that has been accomplished by the time of any particular review. Because of the problems cited previously, though, this typically will be far less than initially intended. The documentation will tend to be too general, and usually there will be little analysis or data available at the time of a review to support the requirements or design being presented.

At this point, the materiel development agency has a problem. The agency will generally not be satisfied with either the content of the documentation or the review, but when they turn to standards such as MIL-STD-490 or MIL-STD-1521A, they will not find the precise, clear guidance or methods to force the contractor to make improvements. When the agency directors consider attempting to do the necessary technical analysis to validate the contractor's work in house, they find themselves lacking in technical staff and schedule time. The only remaining option is to try to "work with" the contractor, using informal mechanisms to solve individual problems, which usually takes time, and this means that data rarely become available much earlier than the contractor would have generated them anyway.

FIGURE 5
Errors in Large Software Systems



This situation is aggravated as time goes on. Because the contractor typically cannot finish his requirements definition and analysis work until well into the design phase, the requirements tend to be unstable, especially early in the design. This phenomenon may be amplified by changes directed by the materiel developer and even sometimes by Congress through funds-cutting legislation. The result is design, redesign, and more redesign, all of which must be documented, all of which cost in terms of dollars and schedule, and all of which ripple through the development cycle.

WHAT CAN BE DONE TO IMPROVE PROCUREMENT?

The areas of highest payoff for reducing risks associated with developing software are activities that occur earliest in the development cycle: project planning and requirements definition.

At the 1977 AIAA Software Conference, Dr. Gansler presented two illustrations of interest to this discussion. Figure 5 shows, based on four major software development programs, that *more* errors in software result from design than from

actual coding. This should be expected, based on the discussion in the last section, and indicates a higher level of technical uncertainty and management stress during the requirements and early design phases. Figure 4 plots the relative cost of correcting a software error as a function of where in the program the error is found. The message is clear. To be most cost-effective, any approach to improving management techniques for software development must be directed at the early stages of the development activity.

The early stages of a development project consist primarily of project planning and requirements definition. Numerous directives developed by DOD have been prepared to help guide these efforts. As an example: DOD Directive 5000.3, as released December 26, 1979, establishes policy for the conduct of test and evaluation in the acquisition of defense systems. Under Section D. (Policies and Responsibilities, Subsection 6.) "Test and Evaluation of Computer Software," specific attention is given to the software components of a defense system and there is a call for participation by the operational test and evaluation (OT&E) agencies in "the early stages of software planning and development. . . ." The directive also calls for "Quantitative and demonstrable performance objectives and evaluation criteria . . . during each . . . phase." Performance objectives and evaluation criteria for each phase of software development are critical in reducing risks. Successful development of software cannot be based on a single, definitive test at the end of the development cycle. Experience has shown clearly that waiting until the final stages of system development to be able to measure acceptability of the software product is disastrous.

On large software development efforts, the development process may extend over 5 to 10 years. This means a high level of personnel turnover and changes in direction from higher authorities may occur before project completion. There is often no concrete, well-definable product until the final stages of development. Documentation and reviews are the primary channel of communications to assess software quality at each phase and must be taken seriously; they must not be allowed to be ignored or minimized.

In order to evaluate the software product at early stages of development, in terms of its functional and performance characteristics, more emphasis needs to be given to simulation. Only when we witness the "execution" of the large, complex software processes that are being built today do we truly understand their behavior and what they will and will not do. When this first execution is the code product, integrated into complicated, extensive hardware subsystems, we incur maximum risk.

Functional as well as analytic simulations at the system and component level can be used to ensure high quality of both requirements and design. Extensive research has been done and is under way to extend the basic concepts of such

simulations, with the overall goal of writing specifications in a language that can be directly translated and executed in a computer. Other areas of research, e.g., the use of metrics to quantitatively measure quality of the software product at each phase, also appear to have future potential in the general context of evaluation criteria.

The DOD Directive 5000.29 makes a thrust in the right direction by calling for requirements validation, risk analysis, and the definition of specific criteria in measuring the attainment of program milestones. In the Army, for example, DARCOM-R-70-16, dated July 16, 1979, "implements DOD Directive 5000.29 by establishing policy and assigning responsibilities for the planning, development, acquisition, testing, training, and support of major and non-major Army battlefield automated systems employing computer resources."

Even with these advances, however, a wide range of interpretations is often applied. The current directives, standards, and guidelines are written for a wide spectrum of applications and are not intended to provide the detailed directives for a specific application. The RFP should be given special attention to ensure that the "essential ingredients" are defined to ensure visibility and control by the materiel developer for the specific development effort, as well as to ensure the quality characteristics required in the software product, e.g., maintainability. Requirements for the competitive contractors to provide software development plans, configuration management plans, design review plans, and quality assurance plans as part of their proposals can help significantly by defining early the detailed "ingredients" of the proposed approach. Typically, these documents are not called for until some months after contract award; i.e., after the software budget has been set in concrete, thus focusing the methodology to be adapted accordingly.

Today, some projects are producing better plans. The major problem is that they often do not appear until well after a project is in trouble. The requirement to submit the plans with the original proposal forces more detailed planning and provides the materiel developer with better visibility into the contractor's responsiveness to the agency's documentation and review requirements. This in turn results in a contract that "from the beginning" calls for documentation, reviews, and audits, not only by name and type, but by the requirements defining specifically what their content will be and the criteria of acceptance to be applied at each phase of development. More realistic bid pricing should also result.

With regard to software requirements definition, a recognition of the role of requirements is beginning to emerge within the entire software development community. Requirements provide the direction that guides every aspect of the design activity and are the standards against which the developed software ultimately

must be validated. A candidate area for reducing risks in today's development is the environment in which requirements are developed and the tools with which they are documented. A careful review of MIL-STD-483 and MIL-STD-490, and a review of the way they are being applied on actual development programs today, suggest that attention can profitably be given to providing alternative guidelines in the area of the content of requirements documentation. These guidelines will help managers, at the outset of a project, clearly delineate the "level of detail and method of presentation" required for the requirements documentation for their particular projects.

Significant activity is currently under way in government, industry, and the universities to define languages for expressing requirements and to develop automated tools to provide management with control and analysis to the completeness, consistency, traceability, and cost of requirements expressed in those languages. The objectives of this research are to provide a means of forcing more discipline on requirements definition, provide a more precise definition of requirements, and to provide management with tools to measure the quality and status of requirements. The necessity for this discipline stems from the environment at proposal time, when decisions are made to cut cost by skimping on requirements documentation and assuming an optimistic test program. Imposing constraints to develop and maintain documentation and techniques for requirements analysis to validate requirements will raise the bid cost of development, because contractors in a competitive environment often do not bid the complete cost to thoroughly document and validate requirements.

The observation can be made that software requirements are derived from system requirements. It follows, therefore, that requirements validation entails a demonstration of some kind that the software requirements are complete, consistent, and traceable with respect to the system requirements and that the performance called for in the software requirements ensures satisfaction of performance criteria in the system specification. Major tools for performing such demonstrations are detailed simulations and models. In a similar sense, the software design can be demonstrated to satisfy requirements using similar tools. These tools need to be planned for from the beginning. In their proposals, competitive contractors should be required to present specific technical and cost data in support of simulation and test facilities to support each phase of development. In addition, those development support and test tools that are deemed useful to maintenance should be required to be formal deliverables to the government.

Another key area of risk is poor reviews at each phase of development. Agendas should be specified by the materiel developer for each review (including requirements, design, and test reviews), specifying in detail the data to be presented at each review. These data include both requirements or design data

and the results of a specified set of test activities. These data should include simulation and/or analysis data for requirements and design reviews, showing explicit predicted performance against the appropriate requirements. At the preliminary and formal qualification test reviews, test data and test data analyses should be presented, demonstrating the performance of the system against all the requirements in the baselined requirements documentation. The agenda should be tailored to the particular procurement and be incorporated in the RFP and contract packages. Some contractual significance should be added to these reviews by also including specific criteria for scoring the contractor's performance at each review relating to the timeliness of completing all review agenda items (including the closing of all trouble reports, action items, and document updates resulting from the review), the quality of documentation as measured against applicable standards, and the thoroughness of data presented. Reviews should be held open until all required data are provided and all problems are closed. Incentive or award fees could be based, at least in part, on review scores.

The need to formulate an RFP that facilitates visibility and control during development activities has been discussed. An eventual contract "with teeth" that forces an orderly, structured approach requiring specified types and levels of documentation and reviews in terms of criteria for their evaluation can lead to more realistic planning on the part of the government and the development contractor.

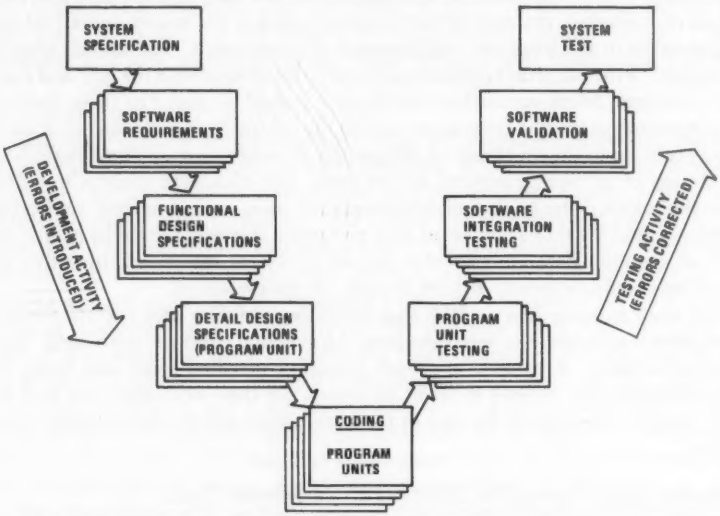
Reducing Risks During the Software Development Cycle

A large number of critical activities, milestones, and software products are required in the development of high-quality military software systems. Visibility into and control of the development activities are essential to both the software developer and the government.

By employing an ordered decomposition of the development cycle, the identification and evaluation of major milestones and deliverables required during each phase can be considered more easily. Each milestone or deliverable can be evaluated within the scope of that particular phase of the development cycle for correctness, consistency, and adequacy. Current DOD directives, guidelines, and standards are oriented toward this approach.

In reality, however, what is often found in many of today's development efforts? A majority of the software problems are being found late in development, during software testing, and not in the particular phase in which they were created. Figure 6 depicts a typical development and test effort. In general, the majority of errors are introduced during development (up through coding), and the majority of errors are discovered and corrected after the product components have been fabricated.

FIGURE 6
Software Development Cycle



Two classes of problems are encountered during testing that need to be given special attention. First, in many applications, a large number of requirements and design errors are not being discovered until late into the testing activities. These errors, present prior to coding, could have been identified and corrected much earlier with significantly less impact on the development activity. Second, coding errors are found late in the system test phase that should have been discovered in a lower-level software test. Both types of problems are symptomatic of rushing to get the product integrated into a total system. This approach, not planned but often followed, is characteristic of a high-risk development effort.

As emphasized previously in this article, it *is* possible to plan and develop software products in which software flaws are identified and corrected early, most often in the phase in which they first occur. For this to be accomplished, however, more analysis and evaluation must take place throughout the overall effort. First, the development contractor must be required to demonstrate at each phase of development the quality characteristics of the product defined for evaluation at that time. Specifics of these characteristics are explored in the

subsections to follow. Second, the materiel developer needs the required resources to assess independently the quality of the software product during each phase of development.

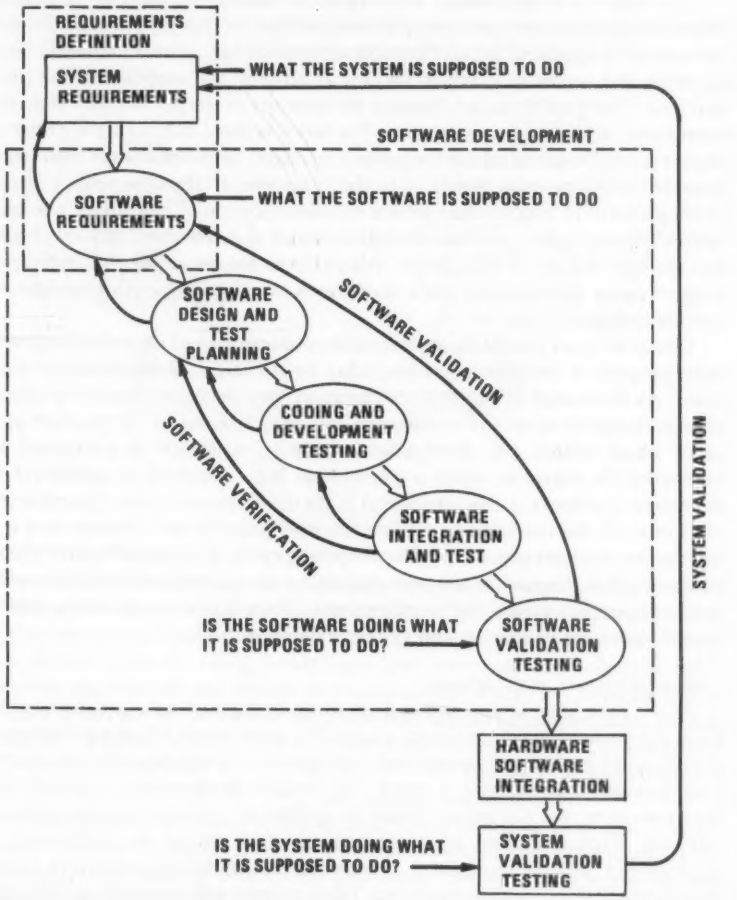
The need for independence with regard to assessment must be emphasized. The materiel developer needs support independent of that provided by the prime contractor. In a general sense, the prime contractor has a conflict between his role as prime and any role in which he acts as advisor and assistant to the project manager. The conflict occurs because the interests of the government may differ from those of the prime contractor. The result is that, although the prime contractor is competent to advise the project manager, he is not always motivated to provide the advice most beneficial to the government. Thus the project manager needs a source of independent advice for identification of risk areas, alternative system designs, quality of documentation, actual status of development progress, and the significance of test results. Alternatives for providing this independent support range from project office staffs or in-house government laboratories to outside contractor support.

One activity of providing an independent assessment of the software development progress is commonly known today by the term "verification and validation." As illustrated in Figure 7, verification may be defined as the evaluation process designed to ensure consistency and completeness of the product at any given phase within the development cycle. Consistency is concerned with measuring the degree to which a given phase (e.g., design) is in agreement with the previous phase (e.g., requirements) in the development cycle. Completeness is a measure of the readiness to initiate the next phase in the development cycle. Validation, the next phase in the development cycle, is oriented toward the final product and is directed at test and evaluation to measure how well the product performs against established requirements. Two types of validation are illustrated: software validation and system validation.

SUPPORT SOFTWARE DEVELOPMENT

In the past, little attention has been given to support software development. From the outset (i.e., procurement phase), the significance of support software as it pertains to both development and maintenance should be evaluated. For support packages that will play a major role in either development or maintenance, the same rigor and discipline should be applied as specified for the application software. Typical support packages that are often critical to development and maintenance include compilers, requirements, simulations, software testbeds (simulating threats, environments, etc.) that support software testing, data reduction packages, and special hardware integration software. As deemed necessary, these packages should be developed as "formal deliverables" and validated to en-

FIGURE 7
Software Verification and Validation


REQUIREMENTS DEFINITION
SYSTEM REQUIREMENTS
WHAT THE SYSTEM IS SUPPOSED TO DO
SOFTWARE DEVELOPMENT
SOFTWARE REQUIREMENTS
WHAT THE SOFTWARE IS SUPPOSED TO DO
SOFTWARE DESIGN AND TEST PLANNING
SOFTWARE VALIDATION
CODING AND DEVELOPMENT TESTING
SOFTWARE VERIFICATION
SOFTWARE INTEGRATION AND TEST
IS THE SOFTWARE DOING WHAT IT IS SUPPOSED TO DO?
SOFTWARE VALIDATION TESTING
SYSTEM VALIDATION
HARDWARE SOFTWARE INTEGRATION
IS THE SYSTEM DOING WHAT IT IS SUPPOSED TO DO?
SYSTEM VALIDATION TESTING

sure the success of the role they will play. The discussion that follows, therefore, is not intended to be oriented exclusively toward the application software, but is relevant to all formal software deliverable items.

REDUCING RISKS ASSOCIATED WITH REQUIREMENTS AND DESIGN

The objectives of software requirements are to provide a basis for the design activities, and to direct the planning for software validation to ensure that the software product operates according to requirements. The requirements must state what the software is to do in terms of inputs, processing, outputs, accuracy, and load. If the requirements do not accomplish these specific objectives, the following problems often occur:

—The software designer is apt to make design decisions that are not in accord with what the requirements writer intended.

—The initial development of test plans is hampered because of a lack of clear understanding of what the software product is to accomplish.

—There is no standard for judging the correctness of test results.

Examples of types of significant problems that often result during requirements definitions are as follows:

—Requirements may be ambiguous.

—Requirements may not be testable.

—The implementation of the requirements may not be feasible.

—Requirements allocations may not be adequately specified.

—Requirements may not be consistent.

When requirements have these characteristics, flaws result in both design and test planning. A detailed independent evaluation of the software requirements (often an iterative process as the requirements evolve) can play a major role in ensuring that they possess characteristics that contribute to a quality product. A representative set of criteria which may be applied during requirements analysis is presented in Table I.

As mentioned earlier, the DOD Directive 5000.29 stresses "requirements validation and risk analysis." Requirements validation may require a wide range of activities, including the conduct of extensive traceability analysis and the development or use of detailed models and analytical simulations to evaluate both functional and performance aspects of requirements.

DOD directives and military standards call for specific formal reviews to assess the quality of the software requirement specification. Formal reviews, however, are often conducted as passive tutorials for high-level management instead of an active evaluation of the specification details. Sufficient time must be scheduled for a detailed evaluation to take place before the formal review. The

TABLE I
Representative Criteria to Support Requirements Evaluation

Realistic - Requirements must be achievable within the capabilities of the data processing hardware.
Unambiguous - Requirements must be stated such that they are definitive and not open to subjective interpretation.
Consistent - Requirements must be consistent with one another, with interfacing subsystems, and with those at the next higher and lower levels.
Necessary - Unnecessary or overly restrictive requirements will increase the cost and complexity of the software and will also impact the design, code, and testing schedules.
Complete - The requirements must completely specify the software product to be provided in terms of accuracy, timing, throughput, interface control, and input/output.

significance of reviews has already been discussed, and it has been recommended that the agendas be incorporated in the RFP and contract packages. These agendas can provide guidance as to the type of documentation and data needed to demonstrate requirements validation. A formal review should be thought of in terms of days, not hours. In addition, active involvement in the formal reviews should be provided by many agencies and groups (including the user, the materiel developer, and the contractor—the systems engineers, software analysts, and test personnel).

By correcting requirement ambiguities, resolving requirement inconsistencies, rewriting non-testable statements so that they possess quantitative characteristics to measure acceptance, modeling requirements, and assessing state-of-the-art technology to ensure feasibility of their implementation, the design and test process will proceed on a more even path. Attaining this goal is an iterative process that requires maintaining the specifications as living documents.

During the software design phase, the most significant contributor to poor quality in the software is often related to the previous phase, i.e., poor requirements. Other problems may still arise, however, because of poor design practices and poor solutions to the design problems. These include poor structure caused by lack of modularity in the design, lack of traceability from design to re-

quirement, erroneous implementation because of inconsistencies resulting from lack of design standards, and mismatched interfaces. Again, the solutions to these types of problems can be provided early in the development cycle through the use of independent evaluation by the government. A detailed evaluation of the design can help to ensure that: Documentation standards are defined and followed to reduce misinterpretation; data flow and control through all parts of the system are adequately defined; emphasis is placed on the hardware involved, data transformation, timing, and the work stations through which control passes; and the design is an accurate representation of the operations and operating rules specified by the requirements.

Design procedures and standards vary from one project to the next. Because of the complexity of the design process, standards and procedures are inherent contributors to the quality of the design specifications and should be required by the materiel developer. Examples of typical standards and procedures that should be considered for incorporation during the design phase include:

—Standards for design documentation format and content;
—Configuration control procedures for requirements and design specifications;
—Procedures to validate design concepts;
—Procedures to support traceability between requirements and design (both directions);
—Procedures for maintaining visibility into, and for the control of, schedules and problem reports.

The objective of an independent evaluation of the software preliminary design is to determine the overall feasibility of the design approach, the capability of the design documentation to support the detailed design, and the capability to test plans to accomplish testing objectives. The preliminary design should be independently traced to the requirements, thus verifying that the requirements have been allocated to the design, and identifying requirements that have been misinterpreted or not addressed. Preliminary test plans prepared in parallel with the preliminary design should be assessed as to their adequacy to accomplish the test objectives.

As the critical design review (CDR) approaches, an independent evaluation can again support status assessment with detailed design verification, test plan and test procedure evaluation, and other support documentation analysis. Functional simulations can be developed and used to verify the functional characteristics of the design. Critical algorithms can be tested using simulation to assess their validity and performance characteristics.

Figures 2 and 3 showed that for large software projects, over half of the defects incurred during development are introduced before the coding activity. When the development activity is decomposed into the three phases of

design/cost/test, a general percentage rule of thumb of 40/20/40 seems to apply when defining the portion of the overall development activity that each phase may be projected to consume. This means that by the time the development effort is 40 percent complete, as many as 60 percent of the defects associated with a large development effort are already introduced into the product. A major risk factor is incurred when these defects are translated into code, and their resolution is left until the test or operational phases.

We often hear that hardware development methodology is far ahead of software development methodology, but why? Hardware developers will not expose themselves to the risks associated with fabricating a product without extensive evaluation directed at removing the design and requirements defects. It is common to proceed in software development without this evaluation.

When software developers started out by developing "small software packages" years ago, they soon learned how easy it was to let the computer help find the defects. Software was easy to change and the design could be documented after the fact if that was required. The usual approach was to "get to the code as fast as possible." This just doesn't work for large, complex software systems. Software is not so easy to change when a large number of personnel are involved. This is demonstrated by the extensive testing required after making a software change to validate that the modification fixed the problem and didn't introduce other problems. It is also demonstrated by the fractured designs and poor-quality documentation (e.g., documentation that does not match the code) often seen at the end of the development phase.

The significance of this problem is supported by the current DOD directives and guidelines calling for extensive reviews and evaluations at each phase of development. The problem is that software developers "try to do too little" and "just get by" without concentrating on the benefits of an in-depth review at each phase. Major milestones are called for during development to examine requirements and design. Risks can be significantly reduced if these milestones are taken seriously and requirements and design issues are identified and resolved prior to the coding phase.

REDUCING RISKS ASSOCIATED WITH CODE AND TEST

The coding phase of software development involves a translation of the detailed design description into a machine-readable form that, when executed, will automatically perform the functions designed. A major activity currently under way in DOD is the development of a standard higher-order language for coding future applications. The DOD Directive 5000.31 specifies currently approved languages for programming. Such a standardization is intended to facilitate both development and maintenance. However, we continue to witness

flaws in software code. Examples of errors that arise during coding that contribute to risks in the development of software include: undocumented differences between design and code, lack of source-code commentary, lack of organization in code structure (usually reflected in the design), and poor configuration management of the code. The last problem results in mismatched code versions during integration and lack of visible control over changes to the code.

Independent evaluation of the code and coding standards and procedures can help to reduce the frequency of these flaws. A major code analysis activity is called for in MIL-STD-1521 as part of the physical configuration audit (PCA). In addition to this formal code audit, selective audits during the testing phase can contribute to maintaining consistency and good quality between the design as documented and as implemented in the code.

The last category selected for discussion is software testing. Actually, this activity may consume 40 to 60 percent of the development effort. Software testing, in this context, is that activity ranging from module (unit-level) testing to total system integration and acceptance. The thrust of this article has been toward doing more early in development to reduce errors so that the testing will go more smoothly, and fewer code corrections will be necessary.

Two specific classes of software tests need to be given special attention—primarily from the standpoint of emphasizing their differences and individual significance. These have been graphically named "glass box" and "black box" testing. Glass box testing suggests looking inside and is what we normally know as development testing. The major characteristic of this testing is that the design is the criterion for evaluating the test results. Special attention is given to data base structures, ranges of values of parameters, and exercising or invoking each instruction and branch statement in the code. Black box testing suggests taking a look at the outside and is what we often identify as validation testing. The major characteristic here is that the requirements are the criteria for evaluating the test results. Both software and system validations are conducted against the software and system-level specifications.

Many questions arise concerning both classes of tests: How much is required? When is it over? Where are we now in the test cycle? These questions reflect the fact that it is often not known how to measure test progress. Techniques have evolved during the past decade to help in this area—some proved and some still conceptual. One proven concept is the use of functional capabilities list to measure progress in development testing and requirements lists to measure progress in validation testing. Such lists, when developed and applied at a detailed level, can be used to decompose a complex, voluminous collection of requirements and design documents into checklists to be used by analysts and managers to guide and evaluate the test process. Not only do nominal processing

paths need to be emphasized, but so do the unexpected or low probability paths associated with contingency processing.

Examples of problem areas that often arise during software testing are as follows:

—Tests are not fully traceable to the software requirements or software design, depending on the class of test under consideration.
—Unnecessary redundancy exists between test cases.
—Critical components are not adequately stressed.
—The required test data are not recorded.
—Collected data from a test are incorrectly interpreted or are not evaluated.
—Test acceptance criteria are too general and subjective.
—Sufficient test support software is not planned for and validated.

These problems contribute significantly to a high-risk program. Most of these problem areas are related to poor test planning.

Four types of documents are usually required for the test cycle: test plans, test procedures, test execution reports, and test results analysis reports. These documents encompass the planning, execution, and analysis of the test process. Each document plays an important role, and independent evaluation during its development can provide benefits similar to those discussed for other development documentation.

A key document in the test process is the test plan. An independent evaluation of a test plan can help to strengthen a weak plan by identifying the weaknesses and recommending enhancements prior to development of the detailed procedures and resources consumption activities (facilities, computers, people, time) required during test execution. A test program will be no better than the plan. Emphasis should also be given to the independent review of test results at each level of software development testing.

Discussed thus far are the types of problems often encountered during testing and how the test-related issues can be resolved early in the test cycle. If effort is expended early in the development cycle to identify and resolve requirements, design, and code problems, the usually complex and difficult job of system/software integration and acceptance testing will be simplified because fewer problems will be found in the product under test. As a result, the maintenance activity will go a lot more smoothly. Up-to-date documentation for both requirements and design will exist, and support software will be available (validated and documented) to facilitate maintenance.

Improved techniques for developing systems have been formulated during the past 15 years, but there continue to be temptations to try shortcuts in attempts to reduce current-year costs. In this sense, software development technology seems to be standing still.

Summary and Recommendations

The preceding sections have examined major risks often encountered in software development, with the objectives of focusing on lessons learned during the past 15 years and recommending alternatives that can result in significantly reducing these risks.

Primary problem areas that lead to schedule slippages, cost overruns, or software products that fall short of their desired goals have been discussed. The software development cycle, often presented as a simple sequence of well-defined phases, continues to be burdened with management and technical problems and issues that work against a high-quality product. Identifying and resolving software problems early, often in the phase in which they first occur, has been shown to contribute significantly to reducing risks in software development.

Major contributors to software schedule and cost overruns have been examined. These contributors are large numbers of requirements and design problems being found after the coding phase and large numbers of coding errors being found during system test activities. The effect of large numbers of fixes after the software package has been coded and integrated into a software system has been examined. Such fixes are very costly and have a deteriorating effect on the quality of the software product. The shift from coding defects to more serious design and requirements issues as a function of the physical size of the software package has been discussed.

The DOD has made advances in outlining what kinds of milestones, documentation, and methodology are best suited for military applications. A wide range of interpretations are often applied to these guidelines, however. The current guidelines are written for a wide spectrum of applications and are not intended to define the detailed directives for a specific application. The RFP (procurement phase) should be given special attention to ensure visibility and control by the materiel developer during software development.

To reduce risks in software development, the potential for the types of problems discussed in this article must be recognized and the necessary steps taken to either prevent their occurrence or correct them as early as possible. In simple terms, a plan must be developed and the necessary action taken to implement the plan. To be effective, the plan needs to be developed as early as possible.

Emphasis has been directed more toward the overall development concept rather than techniques that can be applied when focusing on a particular phase of the development cycle. In summary, the following recommendations are emphasized:

—Formulate an RFP that will facilitate visibility and control during development activities. A contract that forces an orderly, structured approach requiring specific types and levels of documentation and reviews in terms of criteria for their evaluation can lead to more realistic planning on the part of the government and the development contractor, and thus to less risk in the development effort.
—Plan from the beginning for independent evaluation throughout the development activity. This effort will lead to significant flaws being identified and resolved early in the development cycle, flaws that otherwise would lead to major schedule and cost impacts on the development effort.

Implementation of these recommendations will lead to an increased cost projection at the outset. For independent evaluation, a general rule of thumb for cost ranges from 10 to 20 percent of that budgeted by the software developer. However, improved capability to develop software products of known high quality on schedule and within planned budgets resulting from the implementation of these recommendations is the primary goal. It is projected that a disciplined application of these management concepts will result in a reduction in actual system life-cycle costs through reduced development overruns and O&M expenditures.

Today's risks in software development can be significantly reduced through planning and through the application of resources to resolve errors early.||

95

A Software Quality Framework

Dr. Emanuel R. Baker
Dr. Matthew J. Fisher

Computer software is rapidly becoming a major problem for people who develop or use digital computers. Predictions of ever-increasing costs of software relative to computer hardware are coming true.[1] Every day system developers and users are encountering costs and difficulties traced to software that yesterday seemed non-threatening. A recent report by the General Accounting Office (GAO) found that in a number of cases:

> Software development contracts made by Federal agencies have experienced dramatic cost and time overruns, user dissatisfaction, and, sometimes, the contracts resulted in software that never worked in spite of extra time and money spent on it by agency programmers after the contractor had left.[2]

The difficulty is not in the use of computers, because computers by themselves do not solve problems—people do. The problem-solver himself must develop a specific set of programmed instructions that will do the job and generate a usable output. The ability of thousands of programmers and software designers to produce correct software to solve the problem at hand is the heart of the issue. It is, in fact, the necessity to rely on this aspect of software development, i.e., the human endeavor, that makes software development an art. It is the attendant lack of discipline in this "art" that is the core of current difficulties in software.

The Current Software Development Environment

Increasingly, software development is being disciplined and structured by applying new engineering and management principles. We emphasize "new" because direct application of traditional principles developed for hardware has met

1. The term "software," as used in this paper, refers to digital computer software.
2. U.S. General Accounting Office, *Contracting for Computer Software Development—Serious Problems Require Management Attention to Avoid Wasting Additional Missions*, Report No. FGMSD-80-4, November 9, 1979.

Dr. Emanuel R. Baker is the Manager of Product Assurance for the Strategic and Information Systems Division of Logicon. He holds a B.M.E. degree from New York University, an M.S.M.E. degree from the University of Southern California, and an M.S. and a Ph.D. degree in education from the University of Southern California. His research interests lie in defining activities to assess and measure software quality.

Dr. Matthew J. Fisher has been employed by the Department of the Army as a research engineer since 1963 in areas of computer memory technology, navigation, and software quality assurance. He is a member of the Joint Logistics Commanders subgroup for Computer Software Management, which has the responsibility for computer resource standardization policies, regulations, and military standards for the joint services. Dr. Fisher holds a B.S. degree from Drexel Institute of Technology, an M.S. degree from the University of Pennsylvania, and a Ph.D. degree from Drexel University.

with limited success. The reason is that the processes and management of software development are different from those of hardware. The following summarizes these observations:

> Programming technology [or software technology] is now becoming a major science. It is developing as an important physical branch of mathematics in much the same way as physics and engineering. Programming technology has much in common with engineering technology and, in particular, computer system architecture and logic design. It is sufficiently different, however, to require its own disciplines, design practices, theoretical tools, and management standards. The software that is developed as a result of the application of programming technology is of such importance to the performance of any system that significant attention should be paid to the quality of that product.[3]

Keeping these differences in mind, one still finds that the needed discipline for software development may be achieved through the application of a "quality" program. In theory, such a program can promote the establishment of standards and policies to produce quality software, provide continual assessment of the software development, and afford the manager better visibility and control of the software development process.

Unfortunately, what has occured in practice has only introduced confusion. Software quality and quality programs have not been allowed to develop independently of hardware quality or quality programs. They have been force-fit into hardware quality concepts and quality assurance activities and organization. This trend to liken hardware and software quality has brought into software development attendant hardware quality management and engineering principles that are generally inconsistent with software. Recent studies confirm that the hardware quality world, especially quality assurance, suffers from a lack of emphasis, varied implementations, and confusion over terminology such as quality, quality assurance (QA), quality engineering, quality control, product assurance, mission assurance, etc.[4] The relevance of these problems to software development becomes obvious from the content of standards and specifications for software quality assurance (SQA), in the lack of guidance from government agencies, in "force fitting" the SQA role into existing hardware QA organizations, in equating software quality to reliability, or even in equating hardware reliability to software reliability.

---

3. Harold L. Ergott, Jr., "Software Quality Management as a Discipline," Cooper and Fisher, eds., *Software Quality Management*, New York: Petrocelli, 1979.

4. Comptroller General's Report, "Effectiveness of U.S. Forces Can Be Increased Through Improved Weapon System Design," PSAD-81-17, January 29, 1981.

Many traditional QA organizations have been forced by their commands or corporate hierarchy to accept the challenge of software and software quality assurance only because of the organizational title "quality assurance." The differences between commands, or corporate approaches to SQA, immediately surface. Software quality assurance to one company is inspecting labels on magnetic tapes to assure correct tape identification, while other companies exhaust up to 30 percent of project funds on a number of tasks lumped into the name SQA.

Additional confusion stems from a lack of clear policy and direction from the government regarding software quality and software quality assurance. There is no DOD policy specifically covering software quality. Internal government documents, such as regulations, are sparse. On the other hand, the government has published a number of specifications and standards to be applied to industry. These contractual instruments address various aspects of the SQA issue, are inconsistent with each other, are generally ineffective, reflect parochial government viewpoints, and are ambiguous. Clearly, there is no top-down approach.

Outline of Discussion

What is needed is just such a new, top-down approach to software quality, independent of its hardware counterpart and current organizational boundaries. Figure 1 illustrates this approach.[5] In this paper, we will undertake only the first step and explore tutorially the definitions in software quality. These definitions will yield a consistent framework for software quality that places subactivities in their proper perspective within a software development process.

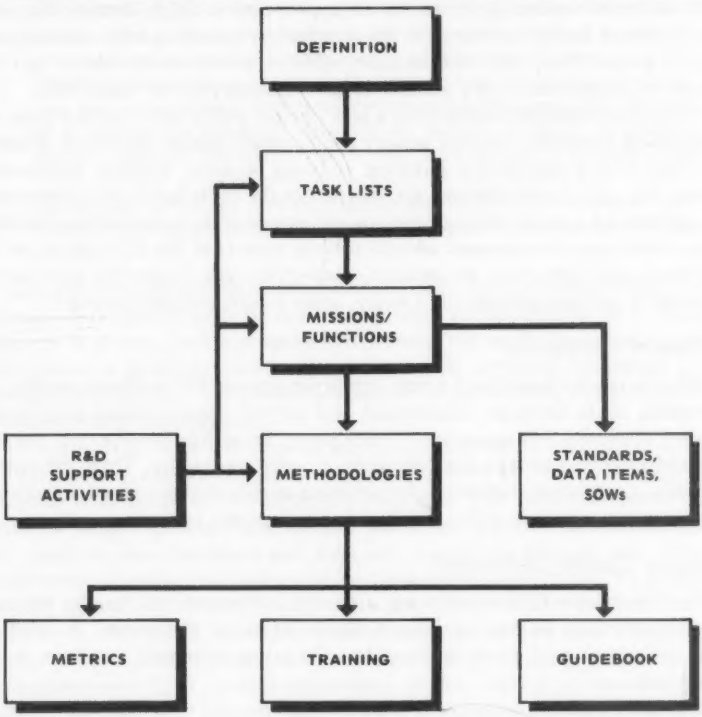
Software Quality Definitions

For the volatile field of software, a quality framework and quality program requirements may be derived from a consistent set of definitions. At least six basic definitions need to be addressed as the prime structural members of our framework:

—Software
—Software quality
—Software quality program
—Software quality assessment and measurement (SQAM)
—Independent SQAM
—Software QA

5. This same approach should be applied to hardware to alleviate confusion in hardware quality.

FIGURE 1
Software Quality Program Top-Down Approach



Other terms may be grouped within these six major areas. In discussing each, we will avoid organizational connotations and relate only to activities and interrelationships.

COMPUTER SOFTWARE

From a software quality and quality program perspective, computer software may be defined as a combination of associated computer programs and computer data required to enable the computer hardware to perform computational or control functions.

Much confusion arises over the definition of computer software because it is abstract and represents the transfer of the problem-solving process from conceptualization to entry into computer equipment, to documentation, or to other physical representation.

Problem-solving may be considered an abstract thought process involving:
—Establishment or formulation of the problem
—A specific implementation mechanism for its solution (e.g., a series of instructions or steps required to obtain the solution)
The final result of this process is a set of instructions (computer program) and associated data that, when interpreted by the computer, produces the desired result. The difficulty in defining software is that the set of instructions is an abstract concept. It can be expressed in human-readable form, such as a source listing or other documentation; it may reside in media that can be accessed by the computer, such as card decks, magnetic tapes, or disks; and it can reside within the computer memory. It is not until the computer follows the instruction set that the software is actually realized. From another perspective, each instruction or series of subinstructions causes the computer hardware to reconfigure itself to perform a specific operation. Each instruction, therefore, produces a different hardware configuration.

Because the set of instructions, or the actions it causes, are abstract, expressing the software for human consumption and assessment is normally accomplished by some form of documentation. From this perspective, one must be concerned with all the planning, rationale, design criteria, standards, and other intelligence involved with the development, operation, and deployment of the computer program and data so that the thought processes behind the problem-solving effort are clearly understood.

SOFTWARE QUALITY

The concept of quality is also abstract. One cause of the problem is that the quality of many items cannot be stated quantitatively, or in a way that allows an objective evaluation. The ultimate result is that communicating the quality of an object to others becomes frustrating, if not impossible, because each individual interprets the quality of the same object from his own perspective. He selects those attributes most significant to him, and rates the item's quality on how well, or to what degree, the selected attributes meet his criteria for excellence.

For our purpose, however, we may define software quality as: The degree to which a software product possesses a specified set of attributes necessary to fulfill a stated purpose.

Notice that this definition applies to the software product, referring to computer programs and associated data. It does not include the associated documen-

tation. In many cases, this deliberate omission causes misunderstanding because the documentation describes the software product. However, the documentation is not the software, but only a description of it. The quality of the documentation is not synonymous with the quality of the product. It should be noted, however, that the quality of the documentation, especially that used during software development, may affect the quality of the final software product.

"Attributes," as used in the definition, includes all specified functional and performance requirements, such as port-to-port timing, data handling protocol, applications functions (trajectory position calculations, guidance commands, statistical estimates, etc.), and other specified characteristics, such as adaptability, maintainability, and correctness. These requirements/characteristics are the determinants of the quality.

Computer software exhibits many attributes that are intrinsic to the software—they exist regardless of what is desired, specified, or measured. They depend only upon the nature of the software. In addressing software quality, it is necessary to specify the important attributes. This specification is required to communicate to others those attributes that constitute the quality. Contractually, this specification is critical.

It is important to understand that the specified attributes constituting the software quality include, as a subset, the desired functional and performance requirements of the software. Many management and engineering activities are directed toward achieving, and then determining, the compliance of the software product with these requirements. (For military procurements the requirements sections of military specifications are, in reality, quality requirements.)

The word "degree" adds another aspect to the definition of quality. "Degree" assigns to the specified attributes the level of attainment with respect to each attribute. The degree or level of attainment may not be quantifiable and, therefore, not measurable in all cases or for all attributes. However, the level of attainment may be assessed; that is, one usually finds difficulty in expressing the desired level or degree of excellence in quantitative terms, e.g., metrics. In many cases, one then resorts to making subjective or non-quantitative judgments of this level. For the functional attributes, e.g., trajectory calculations, the desired degree of attainment is usually expressed quantitatively. An example of such an expression is the following: Calculate the trajectory velocity every second with an accuracy of $\pm$ 2 feet per second for each of the three velocity components in earth-centered coordinates.

In this form, the compliance or degree of compliance of the software has an easy measure; the software either meets the specified requirement, or it does not. For other quality attributes, the desired degree of attainment, excellence, or com-

pliance may not be expressed in such a quantitative form, if it can be expressed at all.

Finally, one must realize that software quality results from activities associated with a software project. Everything that occurs within a software project during development or maintenance affects some attribute of the software and, therefore, the total software quality—used here in its broadest sense. However, all possible attributes may not be of equal relevance, especially when the desired attributes are specified. Furthermore, all actions may not affect the specified attributes and, therefore, the specified quality. In either case, software quality is affected by activities such as requirements definition, the design, coding, testing, and maintenance of the software, activities associated with the quality program for the software, and the interaction of these activities.

SOFTWARE QUALITY PROGRAM

The software quality program is the overall approach implemented to influence and determine the level of quality achieved in a software product. It can be defined as the totality of activities performed by the government, contractor, or their agents to:

—Establish requirements for the quality of a software product;
—Establish and enforce procedures to develop and maintain the software;
—Establish and implement procedures to evaluate the quality of a software product, and to evaluate associated documentation, processes, and activities that impact the quality of the product.

From this definition you can see there are several aspects to the software quality program. One is establishing the requirements for the quality of the software product(s) to be produced on the project. This includes establishing requirements for changes to the software product after it has been placed under some form of formal change control. When the product is changed, the same software quality requirements apply unless these requirements have also been formally changed.

A second aspect pertains to the development and maintenance activities for the software product. Development includes the requirements definition, design, coding, and test phases of the project, and the documentation produced during these phases. Development procedures refer to those needed to define how to design, code, test, and document the software. Examples are procedures implementing top-down design, the use of software development folders, and coding standards to be implemented. The maintenance activities pertain to those activities that occur after an internal (contractor-controlled) code baseline has been established, or after acceptance testing has been completed and the software enters the site installation/operational use phase (government-controlled

baseline). If adequate procedures have not been established to handle the maintenance, the quality initially built into the software product can be degraded as a result, for example, of inadequate change control procedures, or inadequate definition of procedures to "repair" the code in the event of a failure.

The third aspect pertains to those activities necessary to evaluate the software product and related documentation, processes, and activities to determine if it meets established quality requirements. This will be discussed more thoroughly in the next section.

The software quality program covers both technical and management activities. For instance, enforcing procedures for software development is a management activity. The software quality program covers the performance of contractors, subcontractors, and support contractors (such as system engineering/technical assistance, independent verification/validation, and government agencies acting on behalf of the procuring organization).

SOFTWARE QUALITY ASSESSMENT AND MEASUREMENT

Software quality assessment and measurement (SQAM) refers to those activities that determine if the software product meets quality requirements. It may be defined as a set of assessment and measurement activities performed throughout the system acquisition life cycle to evaluate the quality of a software product and to evaluate associated documentation, processes, and activities that impact the quality of the product.

Assessment pertains to qualitative evaluation, while measurement pertains to quantitative evaluation.

The "set of assessment and measurement activities" refers to the actions, such as reviews, evaluations, test, analyses, and inspections, that are performed to determine that technical requirements have been established and that products and processes conform to these established technical requirements, and to determine software quality. They vary as a function of the phase of the system acquisition life cycle, may be performed by varying assigned organizations either within or outside a contractor's organization, which may, or may not, be the same organization that originally performed the activity, process, etc., being assessed. These activities are generally documented in program/project plans, software development plans, project-specific software quality procedures, quality plans, or software quality procedures/manuals.

What is being evaluated is the software product itself and related documentation, processes, and activities. As indicated previously, the "eyes" into the software product are through the related documentation. Furthermore, the processes and activities being performed on the project also provide insight. To illustrate, specifications record the requirements imposed on the software design and

development effort and record the resultant design implementation. A review of the specifications will reveal if all the requirements have been translated into design and if the design is responsive to the requirements. The specification review, when coupled with a review of test plans and procedures, will reveal if the requirements, which have been translated into design, will be appropriately tested.

Processes and activities such as the design and test activities are the second set of "eyes." For example, if the designers are using software development folders (SDFs), the likelihood of achieving the required level of quality is enhanced. Using methodology like the SDF imposes order in the development process, leading both to improved software quality and an improved capability to evaluate that quality. Evaluating a project's SDFs, then, becomes a means of assessing the process of translating requirements into design. Monitoring the test activities also provides the capability to evaluate the level of quality achieved during the design and coding phases of the project.

The bottom line is this: It is the software product itself that must possess the quality attributes. But software *per se* is abstract. What must be evaluated to determine the level of quality attained at any given point in the acquisition life cycle are the related documentation, the related processes and activities, and the results of operations performed by the software during testing of the software product.

INDEPENDENT SOFTWARE QUALITY ASSESSMENT AND MEASUREMENT

Independent software quality assessment and measurement can be defined as the performance of software quality assessment and measurement by an organization different from the organization that produced/performed the products, documentation, processes, and activities being evaluated.

The evaluation action in this case is an independent determination of the state of the software project or products. The evaluation compares the project against pre-established criteria. The personnel performing the evaluations, being independent, are not those who developed the software being evaluated, and may not have established the criteria against which the project is being gauged. (The notion of independence is applied to reduce errors resulting from oversight due to extensive familiarity with the product being evaluated.) This is not to say that these people do not require great technical competence. The task of generating clear, cost-effective methods of implementing such evaluations requires creativity, and is one of the more technically challenging aspects of the software quality program.

Independence, as a concept, has two aspects. One is independence exercised within a contractor's organization, such as through the use of a test team comprising individuals different from those who designed/developed the code. The second is independence within a project structure, where project structure is defined to include all government and contractor organizations responsible for, or performing on, a given project. In the latter case, independence is exercised through the employment of other contractors or agencies, such as using a validation and verification (V&V) contractor different from the contractor producing the software product.

"Organization different from" implies no preordained organizational charters; i.e., there is no implication that a group within a contractor's organization, such as quality assurance or software test organization, must perform all of the SQAM tasks. This is not to imply that independent SQAM is done entirely within a contractor's organization. Part of the independent SQAM activities may be performed by another contractor acting on behalf of the government, or may be performed by another government agency.

An activity within independent SQAM is the collection of objective evidence that technical requirements were established, that products and processes conform to technical requirements, and that software products meet the specified quality requirements. This may mean that one organization does a specific measurement or assessment, but another organization establishes the criteria for the measurement/assessment, verifies that the measurement/assessment has been performed, and impounds the data for eventual use in certifying the product or service. "Objective evidence" includes such items as audit reports, certified test data sheets, V&V reports, and resolved software trouble reports.

The point stressed here is that the activities described as software quality assessment and measurement are very often performed as an independent activity that can take many forms: independent organizations within a contractor's structure, independent agency within the government different from the procuring agency, and/or independent contractor acting on behalf of the government. Decisions about the application of independence, the degree of independence to apply, and the types of independent agencies to employ are a function of many variables. Some key variables entering into these decisions are the size and complexity of the software project, the terms and conditions stated in the contract, the contract statement of work, and the military regulations imposed on the procuring agency.

SOFTWARE QUALITY ASSURANCE

Historically, software quality assurance (SQA) has been interpreted by some to mean the activities necessary to assess and measure the quality of a software

product. Software quality assurance has also been identified with a specific functional entity within a contractor's organization, or within a government agency.

There is a stigma associated with traditional quality assurance. This stigma is in the form of perceptions, often negative, of the quality assurance role for hardware being transferred *in toto* to the software world. By extension, these perceptions influence the perception by software developers of the benefits to be accrued from the application of quality assurance, and the perceptions of the caliber of quality assurance personnel assigned to this role. Compounding this problem is the difference in approaches to SQA from company to company, or government agency to government agency.

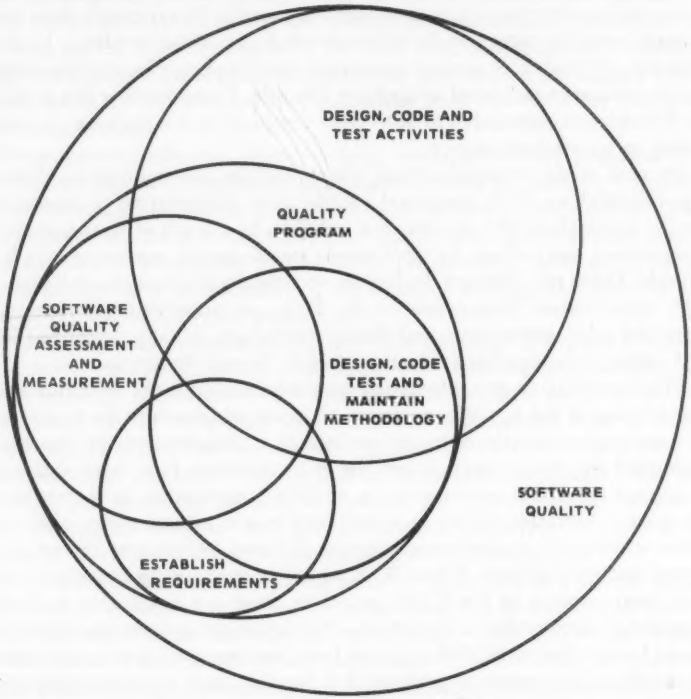
Because of the perception-based stigma and the non-uniform assignment of responsibilities to SQA functional entities from organization to organization, conflict arises about the performance of SQA. As a result of the organizational connotations, very often the implementation of quality assurance falls by the wayside. Either the software developers successfully promulgate an image of an SQA organization ill-equipped to do SQA, or organizational charters circumscribe what assessments and measurements are actually performed by the SQA entity, leaving other evaluations to fall through the cracks.

What is crucial to any software development project is the definition and implementation of the activities necessary to assess and measure the quality of the software products produced by that project, in accordance with the requirements established for that project. When the SQA activities have been defined, the assignment of these activities to a specific organization is a management prerogative. Software quality assurance may be a functional entity within a corporate structure or a government agency. It is not, as has been shown, a well-defined discipline in itself. Where SQA organizations have the capabilitity to perform many or most of the SQAM activities, these can be assigned to the SQA organization (and probably should be). Whichever decision management makes, it must be sure that all SQAM activities have been assigned to an organization or organizations competent to perform that function and, where independence is specified, to an organization with the proper degree of independence.

Software quality assurance and measurement is not synonymous with software quality assurance. Software quality assessment and measurement is a set of activities, and SQA is a functional entity. The degree of SQAM performed by an SQA organization is related to an SQA organization's competence (and independence) to perform those activities.

In today's software world, SQA must acquire the requisite software skills, knowledge, and competence to effectively perform SQAM tasks if an SQA organization wants to be a major factor in the evaluation and certification of a software product.

FIGURE 2
Software Quality Framework



The critical principle is that SQAM activities may be assigned to several organizations. Software quality assurance, being an organizational entity, may be assigned a portion of these activities.

Summary

Figure 2 illustrates the interrelationships of the elements of the software quality framework. In the foregoing sections, we have described these elements and what they comprise. In verbal form, we have attempted to describe their inter-

relations. Figure 2, however, shows with greater clarity what the words may not have conveyed. The key to the implementation of the software quality framework is based on the defintions of the key concepts and their interrelations.

Only the initial step in developing a consistent approach to a software quality framework has been discussed—a set of definitions. Subsequent work will expand upon this set to develop functions that, in turn, spawn various implementation schemes. Similarly, definitions, functions, and implementation schemes allow training. And, last in our approach, we can establish research and development efforts to support our work.

It is important to note that, except for SQA, the definitions that have been developed are divorced from any particular organization. This was intentional. Once the functions are developed from the definitions, the commander or manager has a completely defined set of activities which he may assign to organizations as he deems appropriate. Quality or SQAM efforts, then, are not constrained to any specific organizational entity. And the major payoff is that the required activities will, in fact, be assigned, implemented, and executed, resulting in the production of software possessing the requisite quality attributes to the degree specified.

Software Quality Assurance

108

James A. Dobbins
Robert D. Buck

It is critical to the success of large weapons systems that software be delivered to the field with the minimum number of errors. For this reason, the discipline of software quality assurance (SQA) is receiving increased attention in defense systems acquisition today. Unlike hardware systems, software is not subject to wear or breakage; consequently, its usefulness over time remains unchanged from its condition at delivery. Software quality assurance is a systematic effort to improve that delivery condition.

To provide the highest level of software availability to the field, the entire development process must include quality checks at each step from design through acceptance test. An active program of SQA that identifies and corrects errors during the development process is necessary for an effective software program. This article presents the standards for SQA programs and discusses a program of implementation that offers both management control of the process and early customer visibility.

Standards

Military standards have played an important role in the establishment of software quality programs for defense contracts. The two that seem to have the most impact today are MIL-S-52779(A), "Software Quality Assurance Program Requirements," and MIL-STD-1679, "Weapon System Software Development." Used in conjunction, these two standards will provide the basis for evaluating any program. Familiarity with these standards is a necessity for the contract software program manager and the procuring agency manager.

MIL-S-52779(A) defines the basic elements of an acceptable SQA program and provides customer evaluation criteria. It is the software equivalent of MIL-Q-9858(A). MIL-Q-9858(A) should no longer be applied to software contracts. By definition, an SQA program must be in place and documented, and must address all phases of the software development process. The program must have a method to detect, report, analyze, and correct software deficiencies. The stand-

---

James A. Dobbins is a staff systems analyst with the IBM Corporation in Manassas, Va. He is a specialist in software quality assurance and software reliability and is a visiting instructor in software quality assurance at the Defense Systems Management College. He is also a member of the Talent Bank of the U.S. Congress in the area of software quality assurance. Mr. Dobbins holds a B.S. degree in physics from Loyola University of New Orleans.

Robert D. Buck is manager of the Software Quality Reliability Engineering Department at IBM Corporation, Manassas, Va., a position he has held since 1979. He has held several other management positions on government programs, including Apollo, Integrated Radio Room (IRR) and Safeguard. Mr. Buck has extensive background in software control, software configuration management and software quality and reliability.

ard addresses other elements, but these are the basic elements, without which a program cannot be evaluated.

MIL-STD-1679 is the most detailed of the software development standards and addresses the entire weapon system software development process. Requiring maximum reliability and maintainability of delivered software, it specifies that the software be designed for efficient later change and be accompanied by the related data item description documentation for each delivered software document. The standard for trainer systems software is MIL-STD-1644, "Trainer Systems Software Development," which is virtually identical to MIL-STD-1679.

Procuring Agency Evaluation

Program managers, assigned to develop a system for a limited budget, must have some means available to evaluate a potential contractor. Often the program manager will have little background in software or will have limited time for evaluation. There are alternative resources to evaluate the potential contractor.

One is to seek the services of the Defense Contract Administration Service (DCAS). The program manager must be careful to hire a DCAS organization qualified in software. Many of the DCAS offices have only a hardware orientation. This lack of software expertise precludes effective evaluation of a contractor's SQA program. If the DCAS office is well qualified to evaluate software programs, the program manager will gain substantial knowledge of the contractor's standards compliance. A qualified DCAS software engineer is acquainted with good software practices, aware of the software quality discipline, cognizant of the level of application of this discipline by the contractor, familiar with modern defect prevention and removal processes, and sensitive to cost factors. If a software DCAS facility is not available, alternatives might be another government office familiar with the contractor's quality program, a civilian contractor hired for the evaluation, or a consulting firm with strong software experience. Regardless of the alternative chosen, it is critical that the program manager initiate the evaluation process. The consequences are too important to accept the contractor's claim that a strong SQA program exists.

The purpose of an SQA program is to detect, analyze, report, and correct software errors. If software quality is to be truly effective, the point of application must be examined. Some programs rely totally on test for error detection and removal, while others apply error detection processes from design through final acceptance test.

A program that initiates the error detection process at the start of test will have little ability to recover within cost and schedule if something of significance is discovered. Such eventualities may not hurt the contractor on a cost-plus con-

tract, but the financial impact on the government program manager can be dramatically increased.

Applying error detection and removal processes during the design phase increases the quality of the software at start of test and reduces the cost of error correction. The program manager should examine the contractor's SQA program procedures as presented in the proposal to assure that the error detection process is initiated during design.

Design Inspection

During the design phase of software development, the SQA program should collect statistics on the results of design and code inspections. MIL-STD-1679 specifies design walkthroughs. At the time the standard was written, walkthroughs were the accepted methodology. Since that time, the process has evolved to design inspections. The difference is one of rigor, not intent. The walkthroughs were somewhat informal examinations of a software product by a few technical peers of the product's author. Few records were kept, and no training requirement was placed on participants.

The next evolution of the process was the design review. The review was more formal, kept better records, and gathered some statistical data. Reviews were an intermediary step between walkthroughs and design inspections. The design inspection process is one of considerable rigor requiring participant training. It is the responsibility of the SQA organization at IBM to provide training for the program development personnel.

The inspection process is a peer inspection process performed by a team of programmers who inspect each other's work. The inspection team is led by a moderator trained in the duties and responsibilities of managing the inspection team. Moderators are trained not only in the technical aspects of the error detection and removal process, but also in the psychological aspects of the process. A moderator must be aware of the sensitive position of the author when exposing errors to a group of the author's peers. The aim is to leave the authors with the perception that they have been helped, not criticized. Therefore, all team members, especially moderators, must be trained in the psychological aspects of the inspection process.

During the design inspection process, the moderator provides the inspection materials to the team 5 working days before the inspection meeting. The team examines the material and records errors prior to the meeting. During the inspection meeting, major errors are discussed. Minor errors, such as commentary or spelling errors, are not discussed. A major error is any that will prevent the program from functioning properly. The moderator manages the team's inspection process and follows up on the error corrections after the inspection meeting. If 5 percent

or more of the program design must be changed as a result of the errors detected, the design will be totally reinspected. Otherwise, the moderator will assure that those errors detected are corrected before the program proceeds to the next phase.

All errors are recorded on forms designed specifically for inspections. The completed forms are provided to the author, and a copy is sent to the contract SQA representative. The SQA representative compiles statistics from the error detection forms and from the moderator's summary form. The moderator's form summarizes the errors by type and category, records the man-hours expended in preparation for the inspection meeting, and records the time required for the meeting.

Every element of the system is design inspected. For each design inspection, a portion of the functional specification associated with the program being inspected is included in the preparation materials. In this way, the inspectors determine whether the design will work as written and whether it does what it is supposed to do, assuring conformity to specifications as the system progresses through the development phases.

The SQA representatives collect data on every design inspection conducted. The data is analyzed and compared to expected results based on local norms. The results are also compared to the results derived from other programs being developed. Each month, software quality assurance presents inspection statistics to the software executive management team. The results are examined, and any deviations from expected results are investigated. Once each month, DCAS audits the software quality assurance records and reviews the inspection files during this program evaluation process. In this way, the software development process is visible to the customer without customer participation in the process.

It is important to note that these design inspections are initiated as soon as the design is complete, before the program is coded. As soon as the design is completed, the programmer schedules the design inspection, contacts the moderator, provides the moderator with a copy of the material, and makes any further preparations. This early error detection process prevents any further propagation of the detected errors into the remaining development phases, reducing their impact to zero.

As to the method of design, MIL-STD-1679 recommends strongly, but does not absolutely mandate, the use of a structured design language such as PDL. At IBM Manassas, PDL, which has been modified to match the Ada constructs,[1] is used for all new program designs.[1] The use of a structured design language has significantly reduced the number of errors introduced in the coding process. With

1. Ada is a registered trademark of the Department of Defense.

this in place, a twofold attack on software deficiencies occurs simultaneously—a design language approach minimizing error introduction, coupled with effective design inspection process.

The SQA representatives may participate in the inspection process as moderators or inspectors, but need not necessarily always be present. They do receive the data from all inspections. One of the tasks of SQA is to assure that all programs are design inspected successfully before they proceed to the next phase. Because the program development management schedules the development and publishes those schedules, the schedules are compared to the inspection records to assure that the system is developed in the proper sequence.

Code Inspection

The code phase begins with the completion of the design. MIL-STD-1679 calls for top-down structured programming and defines the specific constructs allowed, which is simply good programming practice. The programs are coded only after they have successfully completed design inspection. Once coded and successfully compiled, they are inspected. The code inspections are virtually identical to design inspections, except that the product is code and the expected errors are somewhat different. The materials for each code inspection include the result of the design inspection. The purpose of the code inspection, in addition to finding errors, is to assure that the code adheres to the design. Often, the coder is not the designer, and creative interpretations might affect the design. Providing the design assures that only the design is coded.

The same data collection forms are used for design inspections and code inspections, providing consistent data collection on every program, at the design and the code phase. Because there are data collection consistencies and expected norms for comparison, early detection of problem areas is possible. This enables the program management team to respond and recover from any potential cost or schedule impacts. Clearly, the earlier problems can be detected, the easier the correction process is likely to be.

An example of the type of report published by the SQA is shown in Figure 1. These data are available to the program management team and to DCAS during their program evaluations. The time expended to perform the inspections is presented on the form. On many programs, there has been a strong tendency to down-load the front-end acquisition and to recover the balance later during test. This was necessitated by some of the single-year acquisition practices in effect on government programs. However, if a truly effective software quality program is going to be in place, it must be activated at the beginning. There is an expression that "quality is free." It may be free in the long term, but it should never be ex-

FIGURE 1

Software Quality Assurance Analysis Report

PROJECT: SASP

TYPE OF CODE: DIAGNOSTICS ____ XXX OPERATIONAL _____

TYPE OF INSPECTION: CODE XXX DESIGN _____

DATE: 2/5/80 DATE LAST REPORT: THIS IS INITIAL REPORT

TOTAL LINES INSPECTED THIS REPORTING PERIOD (NEW, MODIFIED, DELETED): 44445

CATEGORY OF DEFECT REMOVED

TYPE	SYNTAX	COMMENT	DESIGN	TRANSLATION	MISC.	TOTAL
MAJOR (THIS PERIOD)	12	18	18	0	13	61
TOTAL TO DATE	12	18	18	0	13	61
% OF TOTAL TO DATE	19.6	29.5	29.5	0	21.3	
MINOR (THIS PERIOD)	92	238	3	0	0	333
TOTAL TO DATE	92	238	3	0	0	333
% OF TOTAL TO DATE	27.6	71.4	0.9	0	0	
TOTAL DEFECTS THIS PERIOD	104	256	21	0	13	394
TOTAL DEFECTS TO DATE	104	256	21	0	13	394
% OF TOTAL DEFECTS TO DATE	26.3	64.9	5.3	0	3.2	

	MANHOURS FOR INSPECTION PROCESS			REWORK	
	THIS PERIOD	TO DATE		THIS PERIOD	TO DATE
PREPARATION	46	46	HOURS	74	74
INSPECTION	184.8	184.8	LINES	3230	3230
TOTAL	230.8	230.8			

TO DATE EXCLUDING REWORK

MANHOURS PER MAJOR DEFECT IDENTIFIED BY INSPECTION PROCESS: 3.78

MANHOURS PER MINOR DEFECT IDENTIFIED BY INSPECTION PROCESS: 0.69

MANHOURS PER DEFECT (TOTAL) IDENTIFIED: 0.58

DEFECTS IDENTIFIED PER 1000 LINES OF CODE: 8.864

cluded for cost reasons at the front of the development cycle. The benefits to cost, schedule, and maintenance are too great to delay initiation of the software quality program. The alternative is to rely totally on test, and that is courting trouble.

Test

When the Apollo program started, SQA did not exist. The programmers coded and the testers tested. It worked, eventually, and the program did its job admirably—but it was an enormously expensive process. In today's environment, programs with unlimited budgets are scarce. Development efforts must be efficient, effective, and their costs kept at a minimum. This certainly does not mean eliminating test. Testing is very important. It does mean that the test effort must be as effective and as productive as possible. The software should enter the test phase practically meeting specifications from the outset. But if this is so, what would there be left for test to do?

Today's test team is facing a different task than ever before. We are in an age of systems development where combined hardware/software systems are being developed simultaneously, and where the government prefers to assume software maintenance after delivery. MIL-STD-1679 requires a software system to do more than just meet specifications. It will have to pass a third-party stress test where any program stopped for whatever reason means the program is not acceptable. Under these conditions, contractors no longer have the luxury of just meeting the specifications. In order to deliver systems that will truly stand up under battle conditions, we must achieve a quality condition at the start of test that allows the contractor development manager the flexibility to develop a test program to assure that the delivered system meets the expectations of the customer. To have such a quality condition at the start of test, it will be essential to have programs of error prevention and detection in place and working long before the software begins test. The inspection processes achieve such a goal, but there is no room for complacency. The processes now in place must be constantly improved to achieve the ultimate goal, error-free delivery.

Given that the software system has achieved a certain level of quality before it reaches the test phase, the test team should begin with certain administrative tools. There should be a standard method of error recording to facilitate documentation and, where at all possible, entry on a data base. The data base can then be used for tracking the errors, for status briefings to the customer, for priority checks of the errors detected, to track the error detection and correction rates, and to serve as a basis for determining when code may be updated in the controlled libraries.

The test team should, where possible, be a third-party team of professionals other than the programmers. This team should serve as a check and balance on the software developers. In our system, the test team members also serve on the design and code inspection team. The test personnel work closely with the software developers and with SQA. The test team will detect errors, verify fixes, and serve as the first test of the system's success as it approaches the third-party stress test. This is important because that third party, as defined in MIL-STD-1679, is neither the contractor nor the procuring agency. It is, rather, a third party chosen by the customer. There are no restrictions on who that third party may be. It may be a government test team or it may be the contractor's potential competitor. Because a third outside party is to decide if a contractor's product is acceptable, it makes good sense for the contractor to have the best system of error detection and removal possible.

Library Controls

Critical to the success of any test effort, and essential to any truly worthwhile software quality assurance program, is a system of software library controls.

MIL-S-52779(A) requires controls that prevent unauthorized modifications; i.e., the programmers have no access to the controlled libraries, other than to read them. The controlled library system may be manual or automated and must be designed to handle all code to be delivered.

The most feasible system is automated, with the libraries under the control of a department other than the software development department. MIL-STD-1679 also calls for a Software Change Control Board (SCCB), which must authorize changes to the controlled libraries. Various library systems have been devised by many different companies and government agencies, but the most successful seem to be those that are automated and require the final acceptance test to be conducted from the same library used for delivery to the customer. In short, what the customer saw tested is what is delivered. There is no recompile or library transfer following test and before delivery. There should also be a complete documentation of the delivered system contained in a central contractor document control facility. Sufficient information must be contained in the document to enable recreation of the system should the library be destroyed.

Error Slope

It was mentioned earlier that the test data base could be used to track error detection and closure rates. This is one of the most effective tools available to a program manager for determining the state of health of the developing system. If the period of time from the beginning of test and integration (T&I) is determined and the time of acceptance test is known, the cumulative-error profile and the open-error profile should look like the graphs shown in Figure 2. All too often, excessive importance is placed on actual numbers of errors, which is deceptive. An examination of the slope of the cumulative and open curves yields more information. For example, most software development managers can provide a relatively accurate estimate of the anticipated number of errors expected in the software development process if it is similar to a prior project. They know their people and the experience levels. But if the slopes of the curves are ignored, a false impression is obtained. For example, refer to Figure 3. The solid lines are the predicted errors, and the actuals are plotted as points. A program manager might conclude that the system is enjoying an admirable state of good health because the actuals are below the predicted levels. But if the slopes are examined, the opposite conclusion is reached. Because the slopes of the lines are straight, by the time the actuals meet the predicted values (and the manager is thinking that the program is right on target), the situation will be almost out of control. It will take considerable effort to achieve any appreciable curve in the lines, and until there is the required curve, the system will not be ready for acceptance test. Because the

FIGURE 2

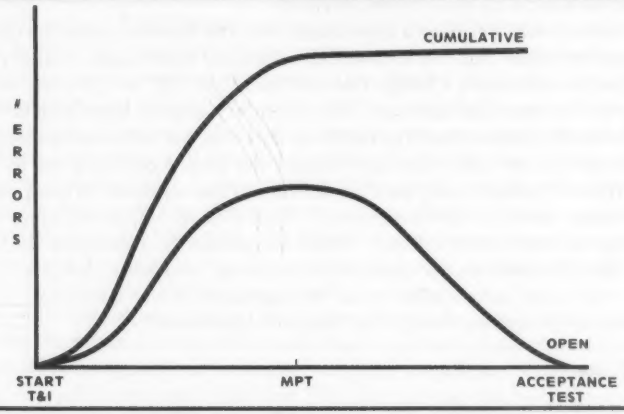


FIGURE 3

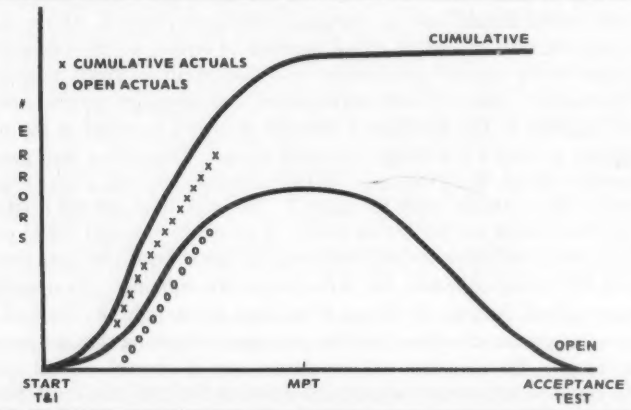
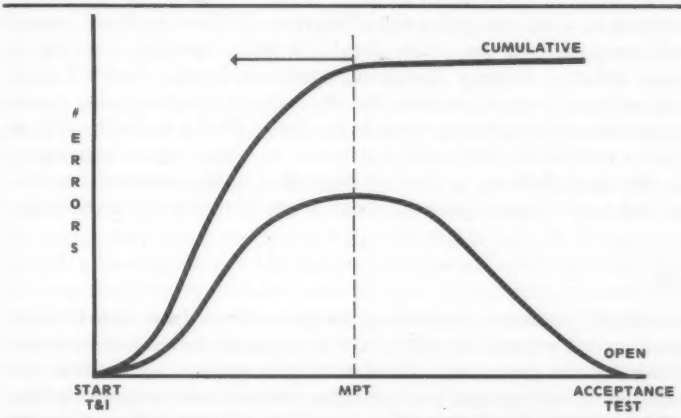


FIGURE 4



schedule dictates the acceptance test date, the program will enter acceptance test anticipating problems. In any well-developed system, the acceptance test should never be a surprise. If the program manager realizes that the actuals have no curve and understands the implication of that condition, there is time to take action without exceeding budget or schedule. Because every system built today for large weapon systems is likely to be composed of a number of major functional components, the first task is to construct a set of graphs of actuals for each of the major functional components. This can tell the program manager which major functional element(s) presents the principal source of difficulty and allows the greatest utilization of available manpower and time to remedy the situation. A subset of graphs should be plotted within each functional element representing each defect priority as defined in MIL-STD-1679. Therefore, within each functional area, there should be five charts, one for each priority. This full set of charts will tell the program manager which functional elements of the system are problem areas and the priority of the errors causing the discrepancy. If the priority is low, the problem is easily addressed. If not, there may be some serious problems which may even require a redesign. Without such an analysis, the program manager is often at a loss to determine the cause or possible solution of the problems. With the analysis, the program manager can anticipate problems and take proper action.

Where We Are Going

If all the available techniques and practices are employed, fewer errors should be introduced during coding, a better quality program should exist at the start of test, dependence on code inspections should decrease, reliance on design inspections should increase, and test teams should be better utilized, resulting in minimal latent defects at delivery. The quality condition shown as the MPT (midpoint of test) in Figure 4 should move to the left, meaning that the quality condition will occur before the midpoint of test in the charts. By the end of the '80s we should expect a compilable design language (Ada), near zero defects at delivery, zero major defects at delivery, a clear understanding of the points of error introduction, and a set of procedures to correct errors at their point of introduction.

Conclusion

Software quality assurance, an evolving discipline, should be expected to continue to develop. It is expected to employ the best possible techniques and practices, assuring that the products delivered as weapon systems are the best that modern software technology can produce. There is no room for anything less. There should be nothing less accepted by any program manager. ||

119 Improving the Combat Direction System Acquisition Cycle

Lieutenant Commander Marvin J. Langston, USN

The age of computers is now. Virtually all new military systems built with electronic subsystems use embedded computer resources (ECR). New systems take advantage of computer software flexibility by using the ECR as the "system integrator." The ECR is used to tie together the diverse sub-elements of a system to achieve a desired overall system goal. This integration function, because of the power of the computer, with its inherent flexibility, is easily performed while preserving some design flexibility for fine-tuning the system. This single attribute, as much as any other, has led to the rapid growth of computers in the military environment.

The DOD Management Steering Committee for Embedded Computer Resources (MSC-ECR) in its June 1979 report stated, "Most development projects are in the unfortunate position of being the first to use one or more (computer-related) technologies. . . . Managers would like to avoid such risks, but the cost of being very conservative and using only proven technology is unacceptable; for example, using computer hardware that is only four years out of date will probably double the hardware cost for a given level of performance. . . . The software technology for controlling these inexpensive computers is advancing at a much slower rate, so software development and maintenance costs will constitute an ever-increasing share of total system life-cycle costs."

Transitioning basic research projects from the laboratory to a system is a multiyear project. During this development time, the leading edge of technology (basic knowledge) continues to advance. This leads to two distinct technology gaps. The first gap is the technology between basic research knowledge and the technology being used in a development project. The second gap exists between the old technology of the present operational system and the technology being baselined for the development project.

The magnitude of the technology gap is a function of both the time required to field a new concept and the amount of activity in related research. In those systems where the military is the primary technology driver, the research activity usually moves slowly. This, in effect, determines the gap between operational systems and the leading edge of technology. In embedded computer command and control systems, the military was a technology driver in the '50s and '60s, but not in the '70s or the '80s to date. Today, the commercial automatic data process-

---

Lieutenant Commander Marvin J. Langston, USN, is Project Officer for the DDG-51 Combat Direction System, Combat Systems Directorate, Naval Sea Systems Command. Previously he was Integration Test Manager for the DDG-993 Combat Direction System at the Naval Sea Systems Command, and Technical Advisor to the Advanced Combat Direction System at the Naval Plant Representative Office, Sperry Univac, St. Paul, Minn. Commander Langston holds a B.S.E.E. degree from Purdue Univeristy, an M.S.E.E. from the Naval Postgraduate School, and is a graduate of DSMC's Program Management Course.

ing (ADP) manufacturers are introducing new hardware, software, and system architectures at phenomenal rates. Because the acquisition cycle for military systems is much longer than that for commercial developments, this rapid introduction of commercial ADP systems inherently keeps commercial systems closer to the leading edge of technology.

Building superior military command and control systems today requires timely acquisition of the latest hardware using well-defined software in an appropriately engineered system architecture. There is an urgent need for those of us involved in ECR systems acquisition to develop methods that will intelligently integrate state-of-the-art hardware with well-designed software. Using advanced development models (ADMs) built with commercial equipment could benefit this process, enhance our system architecture knowledge, and shorten the present system development cycle.

Today's Methodology

An example of a recent ECR system development is the U.S. Navy DDG-993 command and control system development. The combat system for the DDG-993 represents the ECR integration of the missiles, the guns, and the underwater weapon systems with the command and control system.

This system is made up of approximately 320,000 words of on-line computer programs, with another 700,000 words of backup and test programs that reside in the mass storage device. The ship can achieve its full mission capability only when these computer programs and the associated hardware elements function together properly.

The important point here is how the command and control software was developed and implemented. The DDG-993 software was developed using an earlier ship's software as the baseline. All system modifications to accommodate ship design, weapon design, or functional capability were implemented as modifications to the baseline software. Approximately 6 years was needed to build this system from conception to initial operation of the lead ship. There were software changes made to approximately 80 percent of the baseline software, yet the systems remained very similar.

The unacceptable point of this example is that virtually all of the combat system hardware and the baseline software existed in ready-to-use form at the beginning of the command and control design phase, yet 5 years was required to build it into a fielded system. This means that all of the combat system hardware had already completed its 7–10 year acquisition cycle. That hardware was designed with 1960s technology! That hardware was put into a new ship with a planned 20-year life, and it was 15-year-old technology when it left the shipyard. The dilemma is that in ECR systems the hardware is of little value without this

lengthy system integration software design phase. This 5-year system design phase was not the result of poor system acquisition. The fact is that designing software to integrate hardware while accommodating ship variances is a lengthy process. The result is that major changes of system hardware, software, or system architecture are rarely made because of the enormous risk involved. New hardware is usually integrated into an architecture with minimal software changes; or similarily, the software or the architecture is slowly evolved, while holding the other elements relatively constant. It is our inadequate ability to design and field improved system architectures with new software that must be improved.

A Different Approach

The acquisition strategy for ECR systems can be improved to the point that high-quality advanced systems can be provided to the fleet in shorter acquisition times. This can be achieved through risk reduction—risk reduction by the establishment of commercially equipped advanced development model systems. More specifically, this means the establishment of a generic capability for each major type Navy command and control system appropriate to surface, air, and subsurface systems. This effort should be a continuously funded 6.3- and 6.4-level research and development (R&D) project with the established goals of continuously improving the systems architecture and system software, and the system hardware technology generic to all command and control systems.

This system would be built specifically using commercial hardware and software to allow for development concepts that take advantage of the latest available commercial hardware and software techniques. In this manner, the ADM would continually expand its ability to field a broad range of system architecture types using a variety of system software and hardware implementations. The use of commercial hardware also provides a relatively low-cost method of exploring many possible alternatives. New equipment would be added as each new commercial capability is produced that might have application to the generic Navy command and control system. The system software, including the simulation driver, could be written in any chosen language or languages, and commercial software concepts could be applied where they held promise for command and control improvements.

The simulation driver built to support the ADM environment opens up possibilities for commercial-based simulation to support the entire system life cycle. That is, this simulation could be used to support not only the ADM, but possibly the engineering development model and the production training facility.

This ADM environment would have the flexibility to attack the full range of problems from system interconnect architectures to specific software algorithms.

All ADM activity would become knowledge to be applied to the EDM design application. Computer programs might become standardized software modules or tasks that need only be converted to the high-order language of the host system.

Does this idea fit the present environment? Department of Defense Instruction 5000.2 concerning major system acquisition procedures says, ". . . Command and Control Systems require special management . . . management procedures should allow early implementation and field evaluation of a prototype system using existing commercial or military hardware and software."

The Navy is currently working on two related command and control system design efforts. The command and control system for a new destroyer, DDG-51, is the first; the Advanced Combat Direction System, an R&D effort as a base to upgrade carrier and cruiser command and control systems, is the second. In neither case have R&D dollars been spent to investigate system architecture and system software in a commercial-based ADM. In both cases, the next generation of Navy standard computers is scheduled to deliver production units for implementation in the new systems. The AN/UYK-43 is the follow-on computer to the AN/UYK-7, the Navy's large computer, and the AN/UYK-44 is the replacement to the AN/UYK-20, the Navy's minicomputer.

In both these systems, major functional improvements are being installed along with the new hardware. This means that the software will have more risk associated with it. Software modeling work is being done in a suboptimized mode on selected portions of the functional design. No full-system modeling capability exists, thereby increasing the software integration risk. This increased software risk reduces our ability to incorporate new system architecture techniques with the new hardware and software. The result is a conservative approach to system architecture as a risk reducer, while the hardware and software risks are being worked.

Considering the present design environment, this is a rational plan to field the best possible new command and control system in those ships. A capability the surface Navy wasn't well prepared to investigate is distributed system architectures. The DDG-51 design concept called for a ship with increased survivability. The survivability concept may have been best served by distributing the command and control assets into many small computers on a system data bus. The important point is that the Navy did not have enough knowledge or system modeling capability to seriously investigate these ideas considering total system risk. This is unacceptable for the technical needs of the Navy today. The continuous ADM funding in the laboratory test bed could have completely changed the outcome of this example. The scenario could have looked like this:

—Commercial-hardware-based ADM initiated in 1974. Operational with basic capabilities in 1976.

—Capability to model existing CDS systems with multiple processors and multiple computers leads to consideration of distributed processing.
—Using new microcomputers, efforts to model the system with as many as 25 computers in various distributed architectures are investigated.
—The requirements for DDG-51 in 1980 lead to consideration of building a distributed command and control system using the new AN/UYK-44 computers.
—ADM work on solving distributed executive and data base problems successfully leads to a 1983 engineering development model (EDM) using preproduction AN/UYK-44s. The EDM is driven by the commercial hardware and software simulation system built for the ADM.
—DDG-51 is built with a distributed command and control system using AN/UYK-44 computers.

A radical prediction? Yes. But without the ADM concept, it is a prediction that holds little possibility. The present scenario dictates that a system as advanced as that suggested has no possibility for DDG-51 during its life cycle.

Summary

To summarize, I am suggesting that the command and control requirements for the Navy's surface, subsurface, and air systems have generic problems that can be worked and improved upon in a low-cost, technologically expedient environment; that is, a commercial-hardware-based advanced development model. Such a model could establish the entire command and control system capability of today in a 100 percent model driven by a commercial-hardware-based simulation program. Such generic command and control problems as:
—human/system interaction
—information display tailoring
—data base management
—decision aids
—information correlation
should be worked on in the ADM environment. Many solutions could be extrapolated with the latest available commercial hardware and software capability, a capability that is closer to the leading edge of technology. The ADM then would become a library of software knowledge, system interconnection knowledge, interface designs, architecture configurations, and numerous other kinds of enhanced command and control knowledge. This knowledge could then be used in any appropriate way as new Navy requirements and developments are matched with the latest available Navy hardware. Software algorithms might be converted from the ADM software to the Navy standard high-order language (HOL) using a software conversion tool.

Furthermore, if similar laboratory ADM environments existed in subsurface generic and air generic systems, a communication exchange of information for the many system requirements that are similar would enhance the development ideas into the process early enough to allow for new concepts to get into the actual system. In the existing environment, new concepts learned from related Navy efforts can only be applauded. Ongoing development efforts are too firm or time critical to make anything but minor software changes. Total force integration is then impacted at initial operational capability (IOC) or operational testing.

An additional benefit of the ADM if located in the Navy laboratory would be the integrating potential that would exist for all other laboratory work related to command and control. For example, experiments in new display systems, new system interconnects, or new system switching technology would be tested within the larger capability of the ADM. This would have the advantage of validating concepts while acting as a showplace or user test bed of those new ideas.

Today's computer-dependent command and control environment is too critical to allow technology infusion to lag. The commercial-supported ADM offers the advantage of testing concepts, developing system architectures, and system software using technology closer to the leading edge of technology at an acceptable cost. The knowledge base that could be gained from continuous R&D on an ADM would provide seed knowledge for future standard hardware, software, and system architectures.||

125

Evolution of Teleprocessing

Charles K. Watt

The information society is rapidly becoming a reality as micro-electronic computers and other electronic disciplines provide unprecedented universal wide-band networks. With this evolution, the meshing of computer and communication equipment has become so pervasive that communication specialists have coined new words like "compunications," "telematigue," "telematics," and "teleprocessing."

The term "teleprocessing" as used in this paper refers to the combined use of communications facilities and data processing equipment. Dominant among the many elements supporting this definition are technological advances that have tended to blur traditional lines of demarcation between communications and processors, and recent judicial decisions in court cases involving IBM and AT&T.

In this paper, I will attempt to identify the major technology drivers, system relationships, and the trends expected to impact teleprocessing services during the next decade.

MAJOR TECHNOLOGIES

The technologies that have induced major changes in teleprocessing systems and will continue to do so during the next decade are very-large-scale integration (VLSI) circuits, digital techniques, satellites, and optical fibers. Transition of these technologies will continue to take many forms as various elements are designed to provide reliable and timely teleprocessing services in a cost-effective manner over networks consisting of both space and earth systems. The driving force behind this change is the enormous demand for economical information transfer services. The impact of the microelectronics revolution on these services is just beginning to be felt with computing costs dropping sharply. There is widespread utilization of the microprocessor because it is inexpensive and can be programmed to perform an unlimited variety of tasks. This permits more and more software to be written for more and more embedded computer applications. This is the case in both commercial and defense systems, with embedded computers already becoming a dominant element of the total system investment, and software a significant factor in performance.

Charles K. Watt is Deputy Director, Defense Test and Evaluation, in the Office of the Secretary of Defense (OSD). Before joining OSD in January 1981, he held executive-level positions in both government and private industry. His two most recent assignments before joining OSD were Technical Advisor to the U.S. Congressional Office of Technology Assessment and Technical Director of the Naval Electronic Systems Engineering Center, Charleston, S.C. Mr. Watts holds advanced degrees in both engineering and management from The George Washington University and is a graduate of the National Defense University.

Very-Large-Scale Integration Circuits

Integrated circuits, an outgrowth of transistor development and sophisticated solid-state electronics, provide levels of reliability, cost, and capability that are attractive for large-volume, integrated teleprocessing systems of high complexity.

There are a number of developments supporting VLSI circuit fabrication that allow significant increases in the number of devices that can be produced on a single chip. Recent advances include: Hewlett-Packard's 32-bit microprocessor with 450,000 transistors mounted on a single chip; Bell Laboratories Bellmar-32 single-chip microprocessor that dissipates one watt and operates at twice the speed of previous circuits; and Nippon Telegraph and Telephone's microprocessor fabricated with 2-micrometer line widths enabling 20,000 gates to be packed on a chip 12 millimeters square with dissipation of only 750 milliwatts.

As chips get larger and density increases, a new generation of integrated circuits will be possible. One million devices on a single chip is considered to be within reach, and one billion components per chip is considered a future possibility. The memory capacity of a single chip has been quadrupling every 2 years and is expected to continue, with chips containing 8 to 32 megabytes of memory within the next decade. The 64 Kbit random-access memory (RAM) chips are now being shipped in volume. Their small size, high reliability, and low-power consumption make these components suitable for many teleprocessing applications. New logic inside future chips will increase the speed of microprocessors, and the large number of functions on a single chip will greatly increase microprocessor design options. Programmed with high-level languages, systems will feature minicomputer-like architectures and powerful instruction sets that will allow written software to be executed in any location of the microcomputer's address map. By the late 1980s, we can expect to have available a selection of digital signal processing components capable of operating with gigahertz bandwidths.

The large number of functions that will be available through VLSI on a single inexpensive chip implies the existence of virtually free logic and will have important repercussions for microprocessor design. Whereas classical computer theory was developed in an environment in which logic was expensive and communications relatively inexpensive, very-large-scale integrated circuits present the reverse situation minimizing the cost associated with communications paths. This situation is creating a revolution in computer science.

One implication of abundant logic is that designers need no longer be bound to classical architectures, which were necessitated by expensive logic. For example, the memory unit has traditionally been coupled to the central processor unit (CPU) by a communications bus. This design effectively separates memory from the CPU and requires a few bits to be sent to the CPU with each memory access,

thereby wasting access to many thousands of additional bits. Recent VLSI advances permit the distribution of the memory bus over many circuits, effectively giving each logic element its own local memory.

Related to the above is the development of parallel processing architectures. Three possible types are (1) "pipeline" structures where each processing element performs a specific task and the work to be done flows through the processors; (2) processors where many identical elements work simultaneously on different parts of a problem under the control of a single set of instructions; and (3) processors where separate independent elements under separate control structures communicate with each other.

Progress is also being made in the development of complete computer systems in a few VLSI packages with the option of increasing computing power through the addition of modules. Beginning to emerge are micro-computer boards that contain floating point hardware, multiple direct memory access channels, multiple priority interrupt channels, up to one megabyte of random-access memory (RAM), and 64K of read-only memory (ROM). They feature as pluggable options a variety of interfaces to analog signals, peripherals, and communications devices. Programmed with high-level language, they could feature minicomputer-like architectures and powerful instruction sets supporting position-independent code. This would allow written software to be executed in any location of the minicomputer's address map. The re-entrant feature permits interruption of a given program location in the address map, and the return to the proper address from a subroutine. These two design features lend themselves to modular programming, thus making possible "canned software." An applications program can thus be developed by selection of the desired software with progam continuity provided with short, single "batch" programs of high-level instructions. This technique is expected to cut the costs of programming by an order of magnitude. With standard firmware in the form of fully debugged ROM-resident programs, users can build inexpensive software libraries, and split programming tasking into small, manageable modules. Such a simplification is expected to make microcomputers available to millions more potential users.

Digital Techniques

Digital technology owes it existence in large part to developments in solid-state integrated circuits for the computer industry. The pace at which digital techniques are encroaching on the classical analog technology of telecommunications is dramatic. Consequently, it is important to recognize the trends established in digital technology.

The offspring of the marriage between the digital computer and large-scale integration have been in microprocessor, RAM, ROM, and various support chips, e.g., direct-memory access controllers. In the earliest versions, only a low level of integration was possible, and this constrained the architecture to being four bits wide with a modest instruction set and addressing capability. The resulting microprocessor was a programmable logic controller. In later generations, with a higher level of integration, eight bits became possible and a range of capability was implemented from an eight-bit device intended as a powerful logic controller that was approaching the capabilities of the logic and arithmetic unit of a minicomputer. Today, an abundant supply of 16-bit microprocessors with minicomputer-like instruction repertoires are available. Powerful new 32-bit microprocessors are being introduced as "micromainframes" to signify they have the processing capacity for jobs previously requiring 32-bit minicomputers or 32-bit mainframes.

Until these more powerful microprocessors appeared, there was little incentive to program in a high-level language. Applications tended to be modest in size and volume, the desire being to maximize speed and minimize memory requirements. Different assembly languages for different microprocessors met these criteria. With the more powerful microprocessors that extend addressing capabilities to millions of bytes of RAM, high-level languages will be used to a much greater extent. It is not likely that these microprocessors will compete to displace existing minis and maxis; however, Intel Corporation recently developed a three-chip micromainframe with the design emphasis on high-level-language (Ada[1]) programmability. There is also a growing user community that would welcome the capability to transport their software from one microprocessor to another.

There are many applications that strain, and some that exceed, the capabilities of today's largest digital computers. From the beginning, the industry has desired additional throughput. Consequently, we have seen much faster and faster logic and memory components utilized in machine designs and, separately, an attempt to speed things up by great concurrency and computation. In the beginning, the realization was that relatively slow input-output processing was typing up the central processing unit and, consequently, forms of parallel I/O processing were introduced. In some machines the memory was partitioned so that accesses to a single memory were staggered to increase the effective rate of access to the memory as a whole. Pipelining was achieved by overlapping the fetch, interpret, and execute portions of the machine cycle. Finally, computing elements have been grouped together to achieve high throughput rates.

---

1. Ada is a registered trademark of the U.S. Department of Defense.

There will be a continuing need for a hierarchy of storage elements. In addition to the innovative program memories created by high integrated-circuit densities and magnetic bubbles, there will also be improvements to conventional bulk-storage devices, like disks. New disks that provide triple (2.5 billion) bytes, previously available density and capacity per spindle, and even higher-capacity systems are being announced by Tandum (11.5 megabytes), Shugart (10 megabytes), and Irwin International (10 megabytes). Future requirements such as very large archival stores will require further innovation, like video disks.

The man-machine interface is receiving considerable attention as intelligent terminal developments further enhance teleprocessing. The black-and-white CRT with a typewriter-like keyboard and associated printer is being replaced with a flat display panel in color with devices like a light-pen, joystick, and a keyboard for appropriate communications with the system. Keyboards are being human-engineered for the desired set of applications, and associated hard-copy devices now print and plot in several colors. Voice terminals are making considerable progress, and voice-recognition developments are being perfected as a means of computer input. The 16-bit microsystem peripherals are extending the concept of teleprocessing as the swing continues toward distributed processing. Such systems are made possible by dedicated chips designed to be attached to existing processors. With these chips, arithmetic operations can be performed faster and with less program burden.

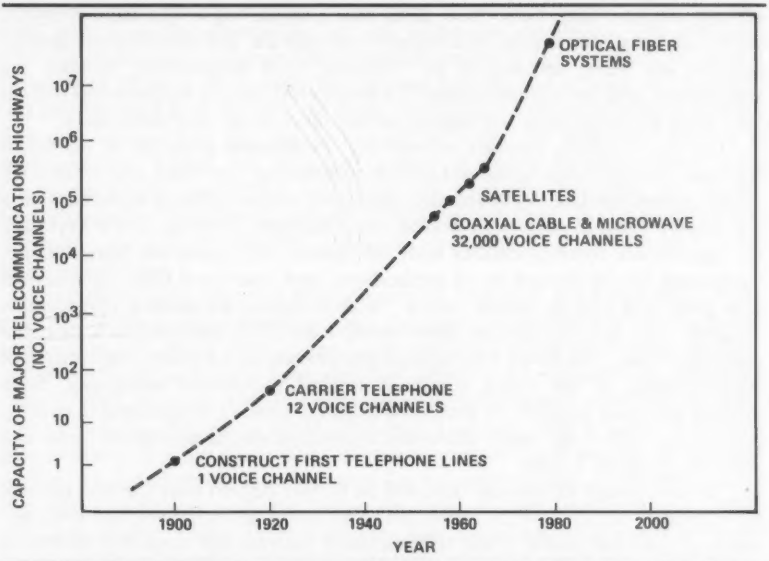
The technology exists now, and will be further refined over the next decade, for intelligent terminals to be connected to networks of computers, and even networks of such networks. These teleprocessing systems will employ a variety of switching and transmission paths and will provide services like integrated voice, data, facsimile, etc.

Transmission Advancements

Transmission technology has advanced during the past century from uninsulated pairs of wires to microwave radio and space satellites. Figure 1 places these technologies in perspective by comparing the tremendous increases in voice-channel capacity. The first systems could carry only 12 conversations on each pair of wires, were vulnerable to interference from weather, and cross-talk between telephone conversations was common. The introduction of insulated wires in 1918, cable carriers in 1938, and the land coaxial cable system in the early 1940s substantially increased both transmission capacity and quality. With 22 coaxial tubes per cable, modern lines can carry up to 132,000 circuits.

About 75 percent of all long-distance transmission in the United States is now provided by line-of-sight microwave radio, which offers lower costs for high capacity. The first system, introduced in 1947, provided 240 circuits. Currently,

FIGURE 1
Telecommunication Evolution



some 1,800 circuits can be provided on each of 11 radio channels of the 1-11 GHz portion of the radio spectrum. Bell Laboratories has developed a new system that will provide 42,000 circuits in the 5 GHz microwave band, and development of the 18 and 40 GHz band is now being pursued to escape crowded conditions in the lower range.

While these technologies continue to provide transmission links for most of the nation's networks, satellites and optical fibers are providing significant revolutionary alternatives for teleprocessing networks.

Satellite links offer high capacity, low transmission errors, and distance-independent transmission paths. Most commercial communications satellites lie in a geostationary orbit. First, coverage is very broad: With an earth-coverage antenna, a single satellite can, in principle, handle users separated by more than 10,000 miles. With the wide-beam satellite antennas in use today, communications are of a broadcast nature with all earth stations capable of receiving infor-

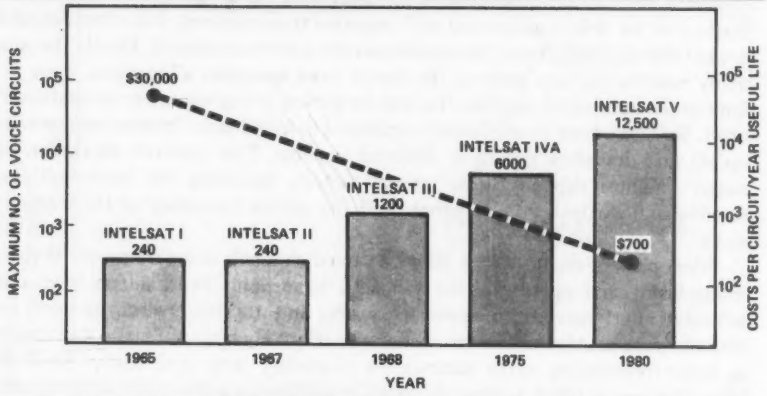
mation transmitted from any other earth station. Second, because the satellite is always in view of the earth stations in its coverage area, expensive tracking equipment is not required on the ground. Third, the round-trip delay to a synchronous satellite has implications for both voice and high-speed data circuits. Because of the delays associated with repeated transmissions, it is often desirable to use forward error correction techniques for data transmission. Finally, because many satellite systems share in the use of fixed spectrum allocations, there is a limit to the number of satellites that can be parked in a given arc or geostationary orbit. Satellites must be sufficiently separated to avoid radio interference between uplink and downlink signals of different systems. This required separation between satellites depends upon several factors, including the beamwidths of satellite and earth-station antennas, and the carrier frequency of the transmissions.

Most present-day satellites offer dedicated channels using frequency division multiplexing and operate in the 4/6 GHz band. This band suffers from considerable interference from terrestrial sources and, therefore, siting an earth station can pose problems. Moreover, because relatively large antennas are required at these frequencies, earth stations are physically large and costly. Once the allocated band is filled, further increases in satellite capacity can be achieved only by re-use of the available spectrum. Re-use is possible by reducing the beam size and increasing the satellite antenna gain so that different beams serve different coverage areas. Capacity can be further increased by polarization diversity, so that two beams with identical or overlapping frequency bands are orthogonally polarized. As satellite capacity is filled, it becomes advantageous to change modulation and multiplexing schemes to improve channel utilization.

In view of the above, satellites are now being developed to operate in higher frequency bands such as 12/14 GHz. The use of the 12/14 GHz band and narrow-beam satellite antennas allows significant reduction in earth-station antenna size and uplink transmitter power, with corresponding reductions in cost. Further reduction in cost and/or improved performance will result from improved transmitter power output stages, antenna beam shaping, and receiver design. Baseband processing data will also be significantly reduced as a result of digital processing and storage advancements. An example of capacity and costs improvements is shown in Figure 2.

However, the increased antenna pointing accuracy and the better surface tolerance required at these higher frequencies serves to offset some of these benefits. Further, an undesirable characteristic of the 12/14 GHz band is that signals can suffer significant attenuation during periods of heavy rain, fog, and mist. Only recently have thorough measurements been made of effects of this attenuation on satellite propagation. Satellite systems using this band must

FIGURE 2
Comparison INTELSAT Capacity and Costs



therefore rely on some form of backup service if 100 percent availability is required at all locations.

The complex problems involved in meeting long-term service requirements include (1) increasing satellite capacity without inordinate increases in mass and cost; (2) increasing system capacity without requiring huge capital outlays for recurring earth station changes; (3) effective utilization and sharing of the orbit/spectrum resource without undue technical complexity and cost; and (4) achieving an acceptable solution to rain attenuation outage problems.

Advances in microelectronics and microprocessors should contribute significantly to needed solutions. Applied to message switching and digital single processing, such technologies are expected to bring satellite-borne operation within economic reach of a mass market. Satellite switching of messages and on-board processing of multiplexed signals could significantly increase capacity over that previously possible with transparent transponders. With on-board switching and processing handled through microprocessors, adapting to shifts in demand with time and place is also possible through power and beam programming (microprogramming of antenna element phase of RF drive), further increasing effective satellite and system capacity. Loads can also be shifted to another satellite. With microwave integrated circuits (MIC), VLSI, and microprocessors, there could be a significant reduction in mass, volume, and, possibly, power drain.

These technologies support integration of many small, switched, multiple, and shaped beams into a single satellite. With the proliferation of point-of-use earth stations, such an approach may become convenient both for satisfying demands for service, and for sharing. Combinations of beam and polarization diversity are already being applied to spectrum re-use, and increasing the number of beams will help. Time-divisions multiple-access (TDMA) techniques may be supplanted by code-division multiple-access techniques as satellites advance to become switchboards in the sky. During 1985-90, fewer, larger, multipurpose, and multiband satellites may begin to replace those of the second generation.

Services should expand to fill the available orbit-spectrum resources. Well-known examples are "electronic mail," multipoint transmission of periodicals, delivery of education and health-care services, computer-computer networking, and organizational and professional teleconferencing. Continued advances in applications of LSI, MIC, and microprocessors, coupled with beam and polarization diversity, may evolve into "orbiting antenna farms" with very large antenna structures supporting many bands. The expanded payload capability of the space shuttle will support experiments with large-scale, deployable structures with very small beams. But the necessity to maximize capacity per unit mass at minimum cost will continue to demand advances in low-mass technologies. New techniques that could change satellites remarkably are optical and acoustic devices for on-board processing and transmission.

Increasing transponder bandwidths will require increasing attention to more linear amplifiers. At high powers and higher frequencies, there will be a requirement for new semiconductor materials and devices with equivalent transmit times appreciably shorter than those now available in the thinnest junctions. In earth stations, there will be a need for low-noise devices at the higher frequencies to keep power per transponder, and total spacecraft power, within reasonable limits for specified performance. Antennas for such stations can routinely be expected to have improved sidelobe suppression. Thus, practical approaches to multiplexing problems could continue to reduce the size and cost of earth station modulation and multiplex equipments.

Finally, technologies of space diversity and new kinds of devices will emerge as higher frequency bands like 17/19 GHz, 21 GHZ, and 30 GHz are explored. The pressures of increasing demand will inspire new techniques that should make these frequencies economically practical for use within the next decade. In the more distant future, laser satellites may prove feasible for transmissions requiring high speed, extreme accuracy, and security.

Optical cable transmission is emerging from the laboratory and shows great promise for reducing costs and making new teleprocessing services possible.

Ultra-transparent glass fibers, a natural transmission medium easily produced for pulsed light signals from lasers or light-emitting diodes, reinforce the trend toward digitized wideband networks. Their first applications in the telephone network are to carry data, digitized voice, and image signals on the trunks between telephone exchanges, where they are cost-competitive with other transmission modes. An especially attractive feature of glass fibers is their ability to carry a large volume of traffic in less space than coaxial or multiconductor copper cables. They do so without causing, or being affected by, electrical interference.

Optical fibers may be divided into two categories, monomode and multimode, each having two distinct modes of propagation. In a monomode fiber, the core diameter approaches the wavelength of the transmitted light and only one mode, the fundamental, is present. In a multimode fiber, higher modes are also transmitted over a core diameter that may be many wavelengths, 50 micrometers or more. Multimode fibers can be either graded index or step index: In the latter, the index of refraction of the core is constant, whereas in the former, it has a definite profile that is shaped to improve signal propagation.

The significance of the modal characteristics of optical fiber transmission lies in the fact that a major cause of signal distortion is pulse spreading owing to the different group velocities of the various modes. This phenomenon is called "intermodal dispersion." The profile of a graded-index fiber is obviously not subject to this phenomenon. The finite spectral line width needed to transmit a given bandwidth gives rise to some material dispersion.

Each of the different types of fibers has advantages and drawbacks. Monomode fibers provide the least dispersion and have a potentially high bandwidth; however, the small size of the fiber core leaves virtually no margin of error in connecting fiber, and limits the efficiency of coupling to the light source. The development of brighter, longer-lifetime light sources and stronger fibers has increased interest in single-fiber per channel technology and several of the major telephone laboratories are developing single-mode, long-wavelength fiber optic systems. Laboratory results have yielded signal attenuation as low as 0.2 decibel per kilometer, and some systems have transmitted data at 140 megabits per second for up to 30 Km without repeaters. Various wavelength-division-multiplexing (WDM) techniques are being investigated to transmit video, voice, and data over fiber-optic telephone systems to the home. Both laser diode and light-emitting diodes at short and long wavelengths are being used in field trials to generate data rates from 1.5 megabits per second to 32 megabits per second. In an altogether different application, Bell Telephone Laboratories plans in 1988 to provide a multifiber cable to undersea telephone transmission that will increase repeater distances to 35 kilometer while increasing two-way voice-channel capacity to 36,000 (coaxial cable now 9.4 Km for 4,200 voice channels).

SYSTEM RELATIONSHIPS

If the aforementioned technologies are to be effectively utilized, they must be integrated into networks. The fundamental technique for accomplishing this is switching systems. Networking is not new. From the earliest introduction of telecommunications, it was known that connecting each terminal with every other with a discrete set of wires was uneconomical, and that some form of central office switching of connections was necessary. Most of the earliest manual switches, with their banks of operator-served plugboards, have been replaced by automatic systems that operate on command of a complex assortment of electrical signals. Fully automatic direct-distance dialing in the United States, an achievement of the entire telephone industry working in concert, exemplifies the scope of planning, engineering, and implementation of networking that is essential to growth in telecommunications capability.

The interaction of telecommunications and computing into a new teleprocessing form of service has placed new demands on the process of achieving point-to-point communications effectively and efficiently. Connection of direct electrical channels between terminals, typical of telegraphic or telephonic services, has been augmented by methods of assigning time intervals, or time slots, on a single channel to subscribers on demand. The time-assignment method of networking is a feature of emerging satellite-based systems and packet switching systems.

Networks

Networks must be able to interact with a multiplicity of voice digitization techniques, digital data techniques, and analog techniques. Transmission rates must be accommodated, ranging from kilobits per second to megabits per second. Services like low-speed video fascimile, now practical almost solely via dedicated circuits, must be integrated on a system basis. Data communication systems must be able to handle interactive, query/response, data base update, and bulk data traffic interchangeably and efficiently. The systems must provide adaptability for the efficient use of trunk capacity for a wide variety of transmission characteristics covering a wide range of transmission rates. They must be compatible not only with established communications links (wire-line, line-of-sight radio, satellite radio), but also with new communications links based on such techniques as time-division multiple-access satellite, and various types of microwave and optical links. They must be capable of maintaining link and large-scale network sychronization with practical networks, incorporating such factors as expandability, modularity, availability, and interoperability. They must be economical in transmission costs by utilizing transmission capacity efficiently. A variety of terminal types will be utilized with transmission rates ranging from hundreds of bits per second to hundreds of kilobits, and transaction sizes ranging from several hundred bits to millions of bits of data.

A number of telecommunication networks with digital transmission and/or digital switching are being developed as information transfer requirements shift from analog to digital data traffic. Technology trends and projections in switching and control must be referred to the existing time and technology baseline. Modern computer-control-led switching and control systems in use worldwide contain sizeable telecommunications installations and are based on technology applications that were in the laboratory-development and field-trial stages about 10 years ago. In other words, developments during the 1960s resulted in full-scale implementation in the 1970s. The introduction of digital switching, which followed the implementation of digital transmission facilities, is occurring in about the same cycle, but on a slightly faster time scale.

Switching and Control

Telecommunication switching has expanded from a simple need of connecting multiple users, to a complex element of public and private telecommunications. The fundamental requirements of a switching system are relatively simple; however, the service and feature requirements are changing rapidly to provide integrated switching on a broad network basis to accommodate multiple and diverse functions. The public switched telephone network historically has used circuit switching where a fixed path was pre-allocated for the duration of a call. This situation is changing with new services offering customer access for terminals and computers. Most radio usage also involves pre-allocation of the spectrum. Another fundamental type of switching system is exhibited where a continuous path is not established, but where memory is used to store the message to be transmitted awaiting the availability of transmission facilities. This form of switching is known generally as packet switching with store and forward switching and dynamically allocated bandwidths. Before the advent of computers, these systems were limited to non-real-time communication, with many manual sorting and routing decisions required along the path of each message.

Because of the expected large increase in telecommunication traffic, the digital networks may be established as the most suitable facility for transmission and switching. Since current systems consist primarily of analog networks, the change from analog to digital must be gradual for economic reasons. In the interim, many alternatives could be used. Digital transmission and/or digital switching will be established in steps, such as digitized junction circuits, digitized truck circuits, and digitized toll circuits. Computer technology has progressed to the point where microprocessor systems are being especially designed for switching and control, and there is no shortage of memory or computing power. As a result of these improvements, networks are rapidly becoming universal translators. The next decade will see a gradual increase in the number of digital systems in service,

and also a start toward combining transmission and switching functions into common systems.

Greatly expanded data communications and information processing techniques are already being applied to enhance the capability of more conventional networks, such as utilization of a few central processors to control the operation of the electronic switching systems in the network. Such systems currently link toll switches and are being extended to integrate local exchange switches in a unified network operations plan. Bell System's latest electronic switching system, the No. 5ESS, makes extensive use of its digital signal-processing chip, which can perform a million arithmetic functions per second-elegant architecture, allowing it to be used in the systems distributed-processing configuration. This chip is organized into three elements that can be controlled independently: a data arithmetic unit, an address arithmetic unit for controlling memory access, and an input/output unit for the serial data interface.

Digital trunk integration is a vital part of the move toward a time-shared digital network. In time sharing, a multiplicity of demand can be met with a unit dividing its operating cycle into time slots in which each element of demand is individually served. Such a time-sharing system approach may be applied to transmission, computation, and switching. Advancements in earth and satellite communications systems are making important contributions to the development of an integrated digital wideband information transfer system, with transmission speeds increasing from a few kilobits per second to hundreds of megabits per second.

Another change that will probably occur in switching networks in the next decade is the incorporation of broadcast satellite features. The ARPA has sponsored extensive research into packet satellite techniques and, during the past few years, has tested these techniques between the United States and England. Packet radio and wire transmission are other areas where ARPA has been sponsoring research and applying dynamic allocation techniques. The basic concept in pack transmission is to share one wide-bandwidth channel among many stations, each of which only transmits in short bursts when it has data to send. This technique appears to be promising for both fixed and mobile local distribution.

A hybrid approach, where both circuit-switched and packet-switched traffic exist concurrently on the same transmission media, is an alternative to a purely packet-switched system. The existing circuit-switched network could be designed so that both circuit- and packet-switched functions are implemented and integrated in a systematic manner. Such an approach may help to solve the fundamental problem of packet delays and overhead characteristics associated with the packet-switched techniques.

SYSTEM TRENDS

General

There is no doubt that advances in microelectronics, very-large-scale integration of circuits, satellites, fiber optics, and many other technologies have begun and will continue to revolutionize the supply of teleprocessing services. The trend is toward more automation in network management. As systems become more distributed in nature, there are more remote facilities with considerable intelligence residing in terminals. The remote facilities tend to be relatively small, sometimes unmanned, so that remote monitoring and control becomes an important part of the overall system. Thus, the overall trend is toward more automation in public and private line network management, with control being centralized or distributed (depending on the size of the network), and with remote monitoring and control. At the chip level, the trend is toward using cheaper VLSI circuits in place of costly software, but some would argue against this trend.

The takeover of many software functions by hardware should help programmers concentrate on development of high-level language programs that are shorter, more efficient, and easier to write, compile, and debut. The original reason for adding high-level language to microprocessors was that in years past memory was slower and more costly than logic circuits in the central processor. This imbalance between memory and central processor no longer holds. Since the inception of ARPANET in 1969, several other packet-switching systems have been implemented. The most notable is Telenet, which has access to international facilities and claims to operate the world's largest data-communications network employing packet-switching technology. For the larger user, packet switching may have a cost advantage over a dedicated private-line network. The trend for packet switching is to integrate voice and data into one network although this would be a prerogative of the "carrier" handling both voice and data, switched or non-switched service. Many local networks provide an architecture for the expanded use of automated office equipment with data rates up to 50 megabits per second.

Residential

The application of broadband teleprocessing to home use is beginning to evolve with the use of TV broadcast and cable television (CATV) for home entertainment. CBS and AT&T plan to test, this fall, a Videotex home information system. CBS will provide data, and AT&T will provide computer facilities and home terminals in tests in 200 homes. There have been other limited experiments and applications, such as interactive medical service and education service, but

there has been no wholesale application. The recent Bell Laboratory development of fiber optic conductors for residential telephones, which will pass a ringing signal that will operate present telephone ringers, provides the possibility that wideband home services could be offered by local telephone companies in the near future.

The British Post Office has been developing for several years a new service known as Viewdata or Prestel service. Several other countries have similar systems under development, and market testing is under way in the United States. Videotex, as it is known here, and Teletext are systems for the mass dissemination of textual and pictorial information for display on low-cost terminals, generally specially equipped television receivers, under the control of the subscriber. Videotex systems operate over normal telephone lines. A computer located at a telephone company central office or other location provides a variety of special services controlled by the subscriber using a keypad or alphanumeric keyboard. Services available may include information retrieval from data bases, interactive functions, electronic mail, telesoftware (the ability to remotely program an intelligent terminal), and electronic funds transfer.

Videotex may also be used with other wideband systems. A keyboard system may use, for example, telephone links for the access of cable television, or broadcast system for the response. Teletext, as opposed to Videotex systems, provides textual and graphic information encoded and continuously transmitted onto the vertical blanking interval of the television signal. Utilizing a keypad, a viewer is able to call up pages of information on a TV screen.

Merging Services

Electronic funds transfer (EFT) services use information processing and telecommunications to transfer value in lieu of processing paper documents. These services may include check guarantee and authorization, credit card authorization, deposits and withdrawals, debits against transaction balances at the point of sale, telephone banking, network banking, and automatic teller machines and cash dispensers. These services, provided by many organizations, are used to transfer funds by government, business, and the public. The EFT services continue to evolve as providers test the marketplace to determine which ones will be accepted by users.

Satellite Business Systems (SBS), a joint venture of IBM, COMSAT, and Aetna Life and Casualty Company, is proposing to provide large capacity, intracompany, digital communication via satellite. These satellites will operate in the 12 and 14 GHz bands. Customer access would be from rooftop antennas located at the customer's facility. The service envisioned by SBS is a specialized, private-

line network, as opposed to a public network. The market for this service appears to consist primarily of large corporations, government agencies, and others with high-volume multi-point communication requirements. Ultra-high-speed switched service will be available to customer-premises equipment.

American Telephone and Telegraph intends to form a separate subsidiary for the future provision of enhanced communicative service and customer-premises equipment. The subsidiary's first offering would be Advanced Communications Service (ACS), a shared data communications service. The ACS would provide a transparent network to promote integration of incompatible terminals and systems in a flexible data exchange system that may be accessed by analog or digital, point-to-point, point-to-multipoint private lines, or by dial access. Using both earth and satellite facilities, ACS may link tens of thousands of corporate and government users across the country.

Services and systems on the horizon, and others like electronic mail and telefacsimile services, will become available. The Postal Service has introduced its electronic computer-originated mail (E-Com), and numerous telemail services are being offered by several private corporations. The "wired-city" concept of everyone using terminals for reading the news, shopping, banking, obtaining education and health-care services, and receiving daily mail, is not necessarily an accurate picture of the near future. However, what is developing more rapidly than most people comprehend is that varied corporations, government agencies, financial institutions, universities, hospitals and other medical establishments, and law enforcement criminal justice communities are using "teleprocessing" routinely. This implies that much of the differentiation that has historically inhibited organization productivity may be neutralized. Certainly more effective organizational integration will be feasible as teleprocessing makes possible new dimensions to accomplish tasks delivered to the desired environment (instead of the inverse). Coupled with the force-multiplier effect of computer power in dealing with highly complex issues, this may be one of the most important contributions this generation will make to near-term productivity improvements. ||

141 Defense Computers and Software—What's Ahead for AI?

Earl C. Joseph

A relatively new technology, artificial intelligence (AI), is rapidly advancing beyond the research stage into practical use. Thus, scientists and managers, as well as those in most other areas of business and society including the defense establishment, could rapidly begin to reap benefits of almost 30 years of AI research. In recent years, AI has been moving toward the development of "expert" systems.

What is an expert system? Today it is a computer system consisting of a set of AI programs that use a stored knowledge base and inference procedures to solve problems. Artificial intelligence research is a subfield of computer science that investigates the imitation of human processes (within computer systems). These AI processes are called heuristics. Heuristics includes learning, symbolic reasoning, logic, inductive discovery and reasoning, deductive analysis, problem solving, and other intelligence processes, including machine representation of knowledge for use in inference tasks. Artificial intelligence assumes such heuristic knowledge is of equal or greater importance than factual knowledge—in fact, for AI purposes, heuristics is assumed to be the process defined as "expertise"—i.e., what "experts" do.

Heuristics goes beyond the use of logical, procedural-oriented strings of instructions operating on streams of data or on data bases, such as occurs in standard computer systems. Simply put, AI heuristics imitates the human brain, especially including heuristic processes for discovering how to solve a problem, or how to diagnose one.

Artificial intelligence expert systems are also known as knowledge information processing systems. Their growth is spawning "knowledge engineering" as a new profession for computer scientists and programmers.

Conventional information processing computer systems execute a string of instructions (the program) as they stream from memory, processing and transforming data in its memory. In knowledge-processing computer systems, overlayered upon the conventional information-processing system, tree strings of knowledge-inference procedures working on data, heuristic rules, and question answers are executed from and on information in memory. Expert systems thread through their knowledge bases via "if-then-and" heuristic rules—that is, the contents of knowledge bases in expert systems is the recodification of knowledge into if-then-and logic. Generation of knowledge bases using if-then-and rules is akin to programming, but without procedure-oriented statements and instructions.

Earl C. Joseph is a Staff Futurist at Sperry Univac, St. Paul, Minn., where he has worked for 31 years in such roles as systems manager, project manager, and staff scientist. He researches the future and advises on future technology, design, impact, application, societal impact, management of future computers, artificial intelligence, and alternative futures. Mr. Joseph holds three computer patents, is the system architect of five major computer systems, and is an Adjunct Professor at the University of Minnesota. He holds a degree in mathematics from the University of Minnesota.

Artificial intelligence heuristics include logical inference procedures that allow semantic access to knowledge bases which use AI processes for making "expert" judgments. The expert systems capture and store the known expertise of a field, and translate such knowledge via AI programs and hardware that offer intelligent assistance to a practitioner in that field—that is, expert systems assist humans in becoming more expert.

Thus, an expert system uses AI inference coupled with a knowledge base for assisting in solving problems, making decisions and judgments, or for creating, discovering, or inventing opportunities. Expert systems allow the tackling of problems that are difficult enough to require solutions that go beyond simple arithmetic or logic, and that require heuristics of significant power for approaching what heretofore required human "experts" for their solution. The knowledge and AI heuristic processes necessary to perform at such an expert level, plus the AI inference algorithms used, can be viewed in the AI expert system as a model of the collective expertise of the best human expert practitioners in that field. Knowledge, once captured in such a fashion in an AI expert system, would also allow a non-expert to apply such expert knowledge and the heuristics to nearly match and often exceed the average unaided human expert in that field. Further, AI expert systems and their knowledge bases can be constantly updated as society gains new knowledge—e.g., "education" of expert systems continues via the updating of their knowledge bases.

What is a knowledge base? The process of building a knowledge base for use with an AI expert system requires the compilation of an extremely "factual" taxonomy of the (each) specialized field—and the heuristics for its application. Such knowledge-based taxonomies turn out to be far more understandable and accurate, and therefore more useful, than today's manuals and textbooks. Since AI systems recently became practical, "knowledge engineering," requiring knowledge engineers, is a rapidly growing new profession. Part of the task is the creation of knowledge bases for use in expert systems. Codifying a knowledge base is a task, as stated earlier, closely related to programming. Today there are few knowledge engineers and perhaps not many that are employed full time. In the future, most computer professionals could become, in one form or another, knowledge professionals. Today, expert systems are computer programs employing artificial intelligence operations using knowledge bases for advising people (in an expert fashion) in the "real-time" of the process of doing something—such as assisting a general, admiral, and other military personnel during tactical situations, or for cooperatively designing a new, highly integrated very high speed integrated circuit (VHSIC) silicon chip, or for assisting decision-making.

Future Alternatives

In forecasting the future of AI expert systems, there are a number of obvious and expanding application areas. Perhaps at the top of the list for the course of future events for the 1980s are AI advice-giving systems for defense. Already expert system programs exist, or are on the computer-aided design (CAD) screens, for medical diagnosis, architectural design, design of very large-scale integrated silicon circuits, molecular generic design, programming, office management decisioning, factory management, home advice (e.g., financial, garden, lawn, repairs) and much more, including many applications in defense systems. Being designed are expert management systems, expert programmer systems, and the like. For example, envision how your profession would be enhanced and changed with an expert defense systems acquisition management system or machine.

Besides consulting, AI expert systems also can assist in the creative, management, design, and other invention arts, as well as giving diagnostic and prescriptive advice. Such dialogues involve the AI expert system threading itself through its knowledge base via "if-then-and" heuristic rules, together with the human it is advising. Some additional near-term uses and markets involve configuration management; robotics; assessments and planning; forecasting; intelligent-agent functions; office automation; image, signal, and voice interpretation; and education and training. Thus, each of these areas, in the future, will undergo drastic changes as AI is implemented in these areas.

Future expert systems, in the form of "people amplifiers" (future remote screen and keyboard computers, or terminals, or hand-held computers), will present factual data or information, and advise or give opinions, based upon the AI knowledge contained in their knowedge bases.

Further, and importantly, an expert system can backtrack to tell the logical process what it went through to arrive at its "expert" advice or opinion.

The knowledge base of an expert system consists of "facts" and heuristics. The "facts" constitute a body, or taxonomy of knowledge (information), that is similar to the information a human expert would use for whatever task such an expert would be performing—codified in if-then-and rules. But herein lies the stumbling block—what does an expert (human) do? Thus, it is no easy task to create a knowledge base that contains expert information and knowledge that is generally agreed upon by experts in a field. Further, not all expert knowledge is a set of "black-and-white" facts; much expert knowledge is codifiable only as alternatives, possibles, guesses, and opinions (i.e., as fuzzy heuristics). Heuristics thus consist of rules of good judgment, fuzzy knowledge, rules of plausible reasoning, as well as hard-and-fast logical reasoning, rules of good guessing, and the like, that are characteristic of expert-level decision-making. Therefore, the perform-

ance level of an AI expert system is primarily a function of the size and capacity of the knowledge base, the quality of its contained expert information, the completeness of its taxonomy, and the number of characteristics of its stored or programmed artificial intelligent heuristics (inference rules and procedures). Today, we are just at the ground floor of creating a variety of knowledge bases. The future will see the variety grow and the knowledge base contents evolve with considerable recodified knowledge.

Thus, there should be little doubt for the future that as AI expert systems evolve to become "experts," humans will possess very powerful people-amplifying tools to assist the defense establishment in almost any task we tackle—including, to reiterate, defense systems acquisition management. In fact, because a knowledge base arranges knowledge in a somewhat procedural fashion, like a computer program, it must be more complete, correct, and comprehensible than the typical textbook or manual. Therefore, experience with current expert systems shows that when compared with traditional sources of knowledge (books, tapes, classrooms, etc.), present and future knowledge-based systems are (or can become) 10 to 1,000 times more complete, precise, correct, and comprehensible.

But perhaps more importantly, as was stated earlier, AI expert systems allow knowledge application in the real-time of human decision-making and actions—to amplify the management process. The expanded use of expert systems can thus be forecasted to affect distributed computer systems development in the following directions: (1) more conviviality—more ease-of-use, friendly and helpful interface functions, (2) more portability for real-time use by managers and professionals, and (3) more inference and knowledge-processing engine architectures (to be acquired and managed by defense system acquisition managers).

Is there an AI expert system in your future? We can now forecast a positive "yes."

Two expert systems recently have reached the "champion" level—they are DENDRAL and MACSYMA. DENDRAL is a knowledge-based expert AI system for solving a class of symbolic chemistry problems. It is the culmination of 16 years of AI research and experimentation at Stanford University. It has reached the level wherein there exists no other better means (human or machine) for doing symbolic chemistry. MACSYMA is a knowledge-based expert AI system for solving symbolic general math equations and systems of equations. MACSYMA solves symbolic algebraic equations and performs differential and integral calculus. It also has reached the champion level at which no other better means (tool) exists (human or machine) for performing general mathematics.

In development are numerous AI knowledge-based expert systems that soon could reach "champion" level status. Included in this growing list are AI knowledge-based expert systems for such applications as air traffic control, computer configuration, VLSI design, crisis management, computer-aided design, and computer maintenance.

Because of these recent AI breakthroughs, funding for AI should be expected to be significantly expanded, especially by the Department of Defense.

The biggest recently announced development in AI comes from the Japanese. In October 1981, the Japanese announced their fifth generation computer for the 1990s. It is to be a knowledge information processing system. Briefly, this Japanese AI computer system, that they have recently committed to development, includes: a knowledge base management system, an intelligent interface system, a problem-solving and inference system, a large knowledge base, a relational algebra machine, an intelligent programming system, an intelligent VLSI CAD development system, and a collection of innovative non-von Neuman mechanisms/processors.

Further, the expected speed ranges up to 1,000 *mega lips* ("lips" stands for logic inference processing steps, and one lip equals approximately 1,000 information processing steps). That is, its upper computational speed range is close to executing the equivalent of a trillion instructions per second! However, future defense systems will need many different types of AI machines, most with lesser capability.

The Japanese goal is to elevate knowledge and information to the level of a basic human need (like food). To do so they realize that they must create a new industry. Their current plan is to leap far ahead of the rest of the world and to lead the world in AI machines. They have initiated this effort. Thus, again the Japanese have set an advanced technology target for us to shoot at and perhaps to go beyond.

The future of AI includes: widening application, rapid growth in usage, growth in AI application programs for assisting most scientific, engineering and management disciplines including defense system acquisition, development of new-inference engine type computers and their acquisition, development, growth, evolution, and acquisition of knowledge bases. Further, AI will transform:

—the future office
—C^3I
—the battlefield and bridge of ships and boats
—the defense establishment

In the new information age, what seems to be emerging is an exponential growth in new knowledge and information, which will require new tools such as AI systems to assist humans in its application. For the AI system to be a useful assistant, this vast growth of detailed information must be reduced into something digestible. Thus, the trend is to squeeze partitionable discipline areas into specific frames, or models. Such reductionalisms are marked by their integrative nature through the universal language of modeling, which is fast taking over the role heretofore played by mathematics. The computer enters the picture as the tool allowing such real-world models to be simulated. Through computer simulation, scientific experimentation occurs and becomes possible without attendant cumbersome laboratory equipment and procedures. By coupling AI knowledge base expertise to assist in the discovery and decision-making processes, future defense system usage of knowledge should be greatly enhanced.

Through modeling and simulation with AI, it is now possible for scientists to "play" serious experimental and mathematical "games" with the object of their research or in tactical situations via the use of inference engines. This is accomplished without first needing to learn sophisticated heuristics, management sciences, laboratory techniques, or mathematics. In other words, the detailed discipline-oriented experimental skills and procedures are embedded within the computerized AI model. Such "robot-simulators" allow decision-makers and scientists to concentrate with the system (knowledge) being investigated rather than being buried within the mathematics and discipline "crafts" to perform the desired experiment or to test a decision. Today, however, we must first learn our field plus the computer simulation/modeling language. With AI expert systems, one need only know (and concentrate on) one's field of expertise. That is, expert AI systems are convivial. Robot AI simulator assistants increasingly will do for the scientist and decision-maker what the calculator does for the average person *in applying arithmetic, such as removing the need to perform* bulky, precise, and rote skill math functions, allowing the researcher to get more quickly and easily to the core matters at hand, i.e., the search and acquisition of new knowledge or for making strategic or tactical decisions.

With future AI robot-simulators amplifying what we do, we could ask questions of computer modeling and have simulated AI experiments performed that otherwise would be nearly impossible or too time-consuming and costly.

Next, envision future robot-simulators in the form of advanced hand-held calculators with voice dialoguing and with AI capabilities. Further, envision them in the form of a simulator for general decision-making rather than for calculating. The utility of such future smart robot simulators for defense system acquisition managers, generals, admirals, programmers, doctors, politicians, voters, and so

forth, becomes obvious when we think of them as people-amplifiers or as "electronic assistants."

Thus, computerized AI expert systems, in the not-so-distant future, could become everyday necessary adjuncts to be worn or carried as part of our dress. They could assist us in many ways, to amplify our minds as well as our bodies, including our interactions and communication with others and with our systems.

Conclusion

For centuries, advances in many different science and technological areas have altered many times over the context of life and the structure of society. Now we are at the threshold of placing society's accumulated knowledge into artificial-intelligence-type expert systems for amplifying what we do with knowledge. Human capabilities are extremely limited when unaided via technics. In the past, mechanistic technological advances amplified human abilities for pounding, locomotion, lifting, toting, cutting, seeing, digging, hearing, resisting diseases, coping with weather, arithmetic operations, and much more.

We are now at a new technological threshold—entering a new era, for amplifying human powers for perceptual reasoning, decision-making, inventing, creating, thinking, and other mental activities. This new era results from our recent acquisition of knowledge for constructing artificially intelligent expert machines for amplifying our minds' activities. Such amplification for raising the productivity of the mind is achieved by imitating artificially the brain's reasoning and other mental powers with computer technology in the form of expert systems.

The result of recent advances in AI expert systems is about to put the defense community at the brink of massive application of artificial intelligence. As a result, a new era is opening that should totally change the character of defense system acquisition, the future hardware and software to be acquired, and much more, including the need to acquire and manage knowledge.

Even though AI has been a subject of study and research within computer science for decades, few are versed in what it is and its new developments. Now, with its recent breakthrough into practicality, the defense establishment is ill prepared for reaping early on its full potential and the opportunities it affords. Nor are we prepared for the changes that it must force as it goes into massive use.

From the foregoing, there should be little doubt that this AI breakthrough should explosively grow in usage in defense systems.

Software Safety Management

David A. Davis
Dr. Nancy G. Leveson

Software safety can be defined as the ability of a computer program and its associated data to avoid system faults that endanger life or property. More specifically, it involves the assurance that software will not cause or propagate failures that result in death, injury, or substantial loss of property.[1] The objective is to guarantee that parts of critical systems dependent upon software control have an acceptable risk level.

The potential for system safety failures caused by software errors is increasing rapidly. Digital computers and their associated software already dominate the automatic control of many system functions. Software monitors and controls missile launch systems, flight controllers, command and control systems, nuclear reactors and other safety-critical systems. More than 80 percent of our weapon systems today require computer software, and this percentage is expected to reach 95 percent by 1985.[2] Failures initiated or propagated by the software in these applications can threaten life and cause damage or loss of valuable property.

In spite of tremendous strides in the development of software engineering principles and practices, programming is still an art form. Furthermore, the inherent complexities are such that error-free code cannot be guaranteed. Although safety techniques have been developed for other disciplines, such as electromechanical hardware, these do not directly apply to the software. The situation is further complicated by the fact that 70 percent of the problems in software systems can be traced back to management and planning deficiencies.[3] Thus, it is important that management give software safety the closest scrutiny and attention.

---

1. N. G. Leveson, "Software Safety: A Definition and Some Preliminary Ideas," Technical Report #174, University of California, Irvine, March 1981.
2. J. G. Griggs II, "A Method of Software Safety Analysis," *Proceedings of the Fifth International System Safety Conference*, Denver, Colo., System Safety Society, Inc., 1981, pp. III-B1-III-B-11.
3. M. Lipow, "Prediction of Software Errors," *Journal of Systems and Software*, 1, 1979, pp. 71-75.

David A. Davis is a systems engineer at Hughes Aircraft Company and is currently working toward a Ph.D. degree in computer science under a Hughes Fellowship at the University of California, Irvine. He has 16 years' experience in the design and development of software for large embedded-computer systems. Mr. Davis holds a B.A. degree in telecommunications from Columbia College, and an M.S. degree in management from Indiana University.

Dr. Nancy G. Leveson is Assistant Professor of Information and Computer Science at the University of California, Irvine. Her research interests are in information system design, software safety and reliability, programming language semantics, and data base systems. She chairs the Irvine Safety Project, which is funded by industry and university grants. Dr. Leveson holds a B.A. degree in mathematics, an M.S. degree in management, and a Ph.D. degree in computer science, all from the University of California, Los Angeles.

This paper introduces software safety from the management perspective. Managerial issues, such as safety policy and safety program planning, are discussed. Software failure modes, hazard identification techniques, and potential correction methods are described, along with a promising software safety analysis tool.

Safety Policy

A policy must be adopted to establish software safety management as an element of overall system safety management. As its charter, it must guarantee the accomplishment of all software-related safety tasks. These include identifying the software safety requirements, achieving identified software safety goals, coordinating with other system elements, and ensuring the effective and timely realization of software safety objectives.

The initial policy issue that management must grapple with is the level of safety required for the system. This is the foremost decision that management must make. It shapes not only the safety program, but the system and software design as well. A line must be drawn between safety considerations and the ability of the system to perform the task for which it was built.

Management must realize that reliability and safety do not necessarily go hand in hand. There is a trade-off between the two that only they can resolve. Reliability is concerned with system performance, i.e., attempting to work around failures to deliver the intended system function. Safety is concerned only with the consequences of failures, attempting to avoid catastrophic situations. This can lead to conflicts between the two goals. Upon detecting a component failure in a weapons system, for example, safety interests might suggest immediate system shutdown. Reliability interests might suggest continuing the mission regardless of greater risk.

Resolution of this conflict rests in a process of negotiation. In order for negotiation to proceed, there must be a specification and division of responsibilities. Thus, one person or group must be designated as responsible for the safety of the software, and this person or group must be independent of and at the same level as those who are responsible for reliability. Negotiation between reliability and safety continues throughout the system life cycle, thus requiring continued resolution by management. It can be most effectively dealt with by defining safety policy as early as possible. Safety policy can then be weighed against contractual obligations, financial requirements, system integrity, and schedule restrictions on an equal basis.

Safety Program

Enforcement of safety policy requires the establishment of a safety program. Safety considerations are planned as a specific responsibility and integrated into all phases of the system life cycle. A disciplined approach is provided by the safety program to systematically evaluate the system design and enforce safety policy. The success of the safety effort greatly depends upon the management activity surrounding a formal safety program. Management provides the emphasis for identifying safety-related problems, isolating their causes, marshaling resources, and initiating solutions.

Successful implementation of the safety program requires the assignment of highly qualified technical personnel. They must be safety-oriented and technically capable of dealing with the complexities of the system. The level of authority they are delegated must be at least equivalent to the disciplines they are required to review. This organizational autonomy guarantees that any severe conflicts encountered will require managerial resolution, and that safety will not be unduly subordinated.

A successful safety program requires sound planning. The plan must describe the objectives of the safety effort and the methods to be used to achieve them. It includes a description of: (1) how the safety function relates to other system functions, (2) the types of safety analyses to be used to identify and evaluate system hazards, (3) guidelines on how the identified hazards are to be minimized and controlled, (4) safety documentation requirements and controls, and (5) the assignment and scheduling of all safety tasks. The safety plan provides management with a means of evaluating the safety effort.

The amount of management attention required to minimize and control hazards throughout the system life cycle is more clearly identified after a preliminary hazard analysis. Conducted during the conceptual phase, this study attempts to determine the hazards that could be present during operational use. All possible hazards are listed and evaluated for their potential causes. Other types of analysis are also conducted at this time to ensure identification of all possible system hazards. When completed, the magnitude and complexity of the safety problems pertaining to the system are discernable to management.

As the system design gains detail, more sophisticated methods of analysis can be employed to trace the causative factors of safety failures. Subsystems and major components can be evaluated. In turn, their components can be evaluated further. Safety critical components can be further and further analyzed to identify the constituent parts whose failure could cause mishaps. For hardware components, probabilities of failure can be assigned based upon experience. Possible design approaches and other corrective action to reduce the failure rate can be ap-

plied. Many detailed analysis methods and guidelines for their usage exist, as do many corrective design techniques. They do not, however, apply directly to software, and management must be cognizant of this fact. One reason is that software failure has different causative factors.

Software Failure Modes

Software does not wear out in the sense that hardware does. Software does fail, however, and does have failure modes.[4] A failure occurs whenever the software does not perform as expected, causing the physical component it is controlling to fail or operate incorrectly. Software can generate erroneous output, causing the premature operation of a component. It can also cause components to operate out of sequence, or it may fail to initiate them at all. Thus, software can and does fail, and it clearly exhibits identifiable failure modes.

Traditional systems safety-engineering methods for analyzing and correcting hazards do not directly apply to software. Different methods must be employed on the software once the traditional analysis has traced safety-critical problems to the software components. To effectively enforce safety policy on software, management must become familiar with software hazard modes and possible corrective measures.

A hazard is defined as a condition that could lead to a safety failure. In software, hazards are caused by errors, which are introduced by human interaction with the software during development or maintenance. This can be through misinterpretation of requirements, mistakes in design, improper code, etc. Completely error-free software is not possible using today's technology. Fortunately, all software errors do not lead to hazards. Software safety, therefore, does not demand error-free software—it only requires the absence of errors that can lead to safety hazards. It is important that management enforce standards and the use of techniques that focus on the prevention of these safety hazards.

Software Hazard Identification

Hazard identification for software begins at the requirements specification stage. Software obviously cannot injure anyone; only the instruments it controls can do this. During the early development stages of a hardware system, a preliminary hazard analysis (PHA) is performed. The PHA is used to determine the potential hazards of the system, and the safety requirements for the software involved are derived from the PHA. The PHA is used to help determine which portions of the system should be implemented in hardware or software. Further-

4. J. C. Thomas and N. G. Leveson, "Applying Existing Safety Design Techniques to Software Safety," Technical Report #180, University of California, Irvine, 1981.

more, for hardware that is to be software-controlled, the PHA suggests guidelines and procedures that must be followed in the software development, and provides important input to the software safety requirements specification process. Management must ensure that the PHA clearly identifies all software safety requirements.

The first step in the production of any large software system is the requirements specification. This specification normally includes the functional requirements of the system. In safety-critical systems, it must include also the safety requirements. It is important to management that the software-requirements specification include not only what the system will do, but also what it will not do. It should include means for eliminating and controlling hazards and for limiting damage in case of a mishap (i.e., the failure of a safety critical function). Fail-safe procedures must be outlined. An important part of the safety requirements is the specification of the ways a system can fail safely, and to what extent failure is tolerable.

Safe Software Design

A safe software design includes not only standard software engineering and fault-tolerant techniques to increase reliability, but also special safety monitoring and fail-safe procedures. Fail-safe design has received relatively little systematic attention in software research.

The goal of fail-safe procedures is to limit the amount of damage caused by a fault. No attempt is made to satisfy the functional specifications except where this is necessary to ensure safety. Procedures where no attempt is made to maintain functionality are appropriate for such applications as missiles and nuclear power plants, where the most appropriate action in case of failure is to simply shut down and do what is necessary to prevent further damage (e.g., disarm a warhead or limit the amount of radiation leakage).

There are some systems where it is not possible to simply shut down in the event of a failure. To maintain safety in software that provides life-sustaining activities, the damaged system must be kept functioning. In this case, the fail-safe procedures must include some fault-tolerant provisions; that is, to continue operation but perhaps to provide only degraded performance or reduced functional capabilities until the fault is removed. Fail-safe components in these situations may also initiate measures to eliminate the possibility of an accident caused by the failure. An example of an application where fail-active procedures are appropriate is an air-traffic control system. In the event of a failure, operations must continue but can do so at a reduced rate or, alternatively, flights can be diverted to other areas.

A safe design must include not only fail-safe procedures, but also facilities for error detection and monitoring of critical parameters. Continual or intermittent monitoring is essential, since under most circumstances, it is important to detect hazardous conditions while there is enough time to prevent a safety failure. This monitoring may be included as an integral part of the software system, or may be in the form of a separate safety monitor. A separate monitor has the advantage of allowing continual monitoring and immediate interruption of normal processing in the event of the detection of unsafe conditions.

Some other aspects of a safe design are separation of critical and non-critical functions; on-line reconfigurability and fire walls so that errors in non-critical functions will not affect critical modules; and the use of operating systems techniques, such as capabilities or monitors, to limit access to safety-critical data and modules. It is important that errors in non-critical routines not be allowed to affect critical functions. Of course, the use of software fault-tolerance techniques, such as n-version programming or recovery blocks to increase the robustness of the software system, is also important.[5] What is needed is an extension of these techniques to include safety considerations. Work on this problem is currently in progress at the Irvine Safety Project.

Software Safety Analysis

Once the safety requirements have been specified and a design proposed, it is necessary to analyze the design's safety and, later, to measure how well the completed system satisfies the safety requirements. It is management's responsibility to see that this is done.

While designs can be analyzed for various qualities, a safety analysis is accomplished by separating the safety and functional aspects of the design and finding failure modes or failure scenarios that are or could lead to safety failures, or, alternatively, showing that the logic contained in the design is not likely to produce any safety failures. The methodology essentially assumes that any procedure, value, or input has the potential to be incorrect. The purpose of the analysis is to determine which incorrect values, functions, or combinations of these will cause a safety failure. Information regarding these possible failure modes can then be used to guide redesign of the the software, to include additional safety failures, or to facilitate thorough testing by pinpointing critical functions and test cases.

---

5. L. Chen, and A. Avizienis, "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation," *Eighth Int. Conf. on Fault-Tolerant Computing*, Toulouse, France, June 1978; B. Randell, "System Structure for Software Fault Tolerance," Proc. Int. Conf. on Reliable Software, *SIGPLAN Notices*, 10:6 (June 1975).

Working on the Irvine Safety Project, Peter Harvey has described a technique for accomplishing this analysis using fault trees.[6] His technique is derived from a similar one used in safety engineering, which was jointly developed by Bell Telephone Laboratories, the Air Force Ballistics System Division, and the Boeing Missile Division, and was originally used to analyze the hardware design of the Minuteman ICBM.[7]

The information produced during the specification, design, and analysis stages as described above influences the rest of the production cycle. For example, the safety requirements and analysis are reflected in the actual code through assertions and safety checks, in the testing and validation stage by directing the testing procedures, and in the procedures for error reporting and control. Probability estimates of safety failures can be determined from the use of fault trees together with reliability estimates of the critical components of the system. These figures are then used to further guide the testing and validation efforts.

Conclusions

Software safety is an important software quality that has great impact on the entire life cycle of critical real-time software. Management is the key element in the production of safe software. Management organizes and provides the funding necessary to establish a software safety program. Management guarantees the accomplishment of all software-related safety tasks. These include identifying the software safety requirements, achieving identified software safety goals, coordinating with other system elements, and ensuring the effective and timely realization of software safety objectives. ||

6. P. Harvey, "Fault-Tree Analysis of Software," Master's thesis, UCI, 1982.
7. R. L. Browning, *The Loss Rate Concept in Safety Engineering*, New York: Marcel Dekker Inc., 1980.

155

Software Logistics: A Sleeping Giant

Paul J. McIlvaine

The irreversible and increasing commitment on the part of the Department of Defense to military systems with embedded computer resources is becoming well known. Nearly all mission-essential DOD operational and management processes are now dependent upon information technology. By 1990, we will see 26 times the numbers of embedded computers that existed in 1980.[1] This major increase is being accompanied by significant cost perturbation—the cost of computer hardware is decreasing, while the cost of computer software is increasing. The implication of this is particularly acute in the arena of software logistics. Some recent rules of thumb are so alarmist as to project that it costs $75 per line of code to develop software and $4,000 per line of code to maintain that same software. These trends call for re-examination of the logistic community's attitude toward computer software, because cost-effective software integrated logistic support (ILS) may become more elusive in the future.

The Two Communities

The cultural impact of computer technology, in which the "state of the art" has advanced in time at a rate greater than any other technology, has been to create two prime factions—the hardware community and the software community. Those who are equally well versed in both hardware and software technology are clearly a minority group at present.

This phenomenon is especially pronounced in the logistics community, where hardware technologists can be seen to predominate. It wasn't until 1980 that computer resources support was identified for the first time as a distinct element of the nine elements of integrated logistic support (supply support; technical data; facilities; manpower and personnel; packaging, handling, storage, and transportation; training and training devices; support and test equipment; computer resources support; and the maintenance plan).[2] This identification, however, fell short of becoming a positive advance when DOD failed to provide a basic definition of computer resources support.

1. "DOD Digital Data Processing Study—A Ten Year Forecast," Electronic Industries Association, 1979.
2. DOD Directive 5000.39, 17 January 1980.

Paul J. McIlvaine is a Professor of Systems Acquisition Management in the Technical Management Department, School of Systems Acquisition Education, at DSMC. Previously, he was Head of the Advanced Systems Unit, Tactical Systems Section, at Headquarters Marine Corps. Mr. McIlvaine holds a B.E.E. degree from Villanova University and an M.S.A. degree from George Washington University.

As pointed out in an earlier article, computer resources support can either be considered as a separate ninth element of ILS or thoroughly integrated into the eight other elements of ILS, the latter approach being preferable.[3] Common policy and practice within DOD, however, tends to favor the former approach. Separate preparation of a hardware-oriented Integrated Logistic Support Plan (ILSP) and a software support plan—such as the Computer Resources Integrated Support Plan (CRISP)—is a good example of this separation. The false solution of physically locating a separately prepared CRISP as an addendum to a separately prepared ILSP may give the impression of integration, but does not truly result in the preferable unified system of hardware/software logistic support.

THE MAINTENANCE PLAN

Maintenance largely consists of tasks and procedures for restoring the operational capability of a system upon failure. Software maintenance, depicted in Figure 1, is more complicated because of the unique nature of the technology. Failures typically trigger the need for corrective maintenance action, but the severity of software failures range the full gamut of mere operator inconvenience to complete program stoppage. Hence, the nature of any software failures provides a mitigating circumstance that must be taken into account. The restoration of the operational capability of the software is also unique in that it is accomplished by means of a software modification, not just the mere replacement of a failed or burned-out part with an identical, new part.

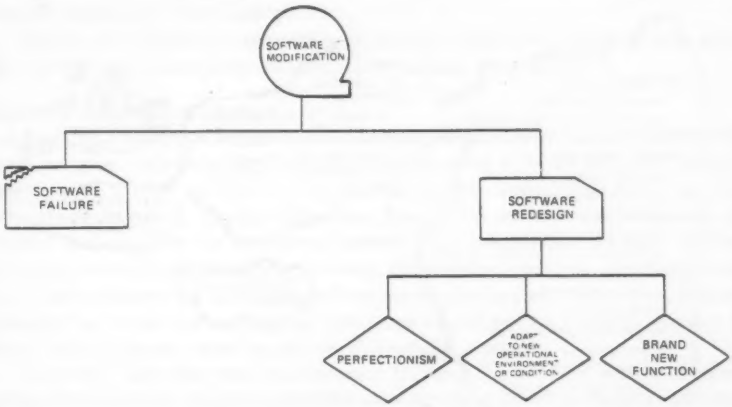
Another complicating circumstance is that software redesigns intended to add new functions, perfect lines of code, or adapt to new conditions are accomplished in the same manner as software maintenance actions to restore the operational capability of the system. Problems in perspective also come into play here. Although the fixing of software failures is clearly the No. 1 priority from a user point of view, software redesigns to add new functions are clearly preferable to the software technologist.

The aggregation of software failures and redesigns over the life cycle of a system is commonly referred to as software maintenance or software support. It normally requires a sustained level of effort over the operational phase of a system's life cycle.

Significant controversy still exists over whether software actually fails. Certainly, it does not physically break or degrade. Software failures are due to anomalies undetected during software design and/or test. Unfortunately, because of the almost infinite variability of input conditions, it is impractical (from a cost

3. Paul J. McIlvaine, "Let's Get Rid of Computer Resources Support," *Logistics Spectrum*, Volume 15, Number 3, Fall 1981.

FIGURE 1
Nature of Software Support



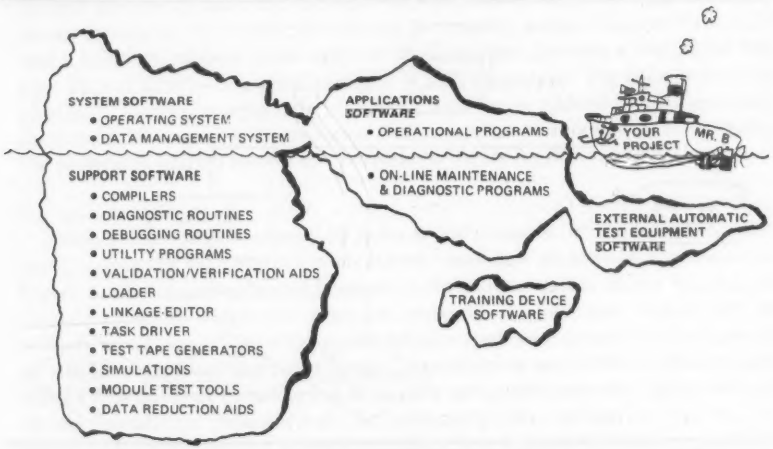
and time standpoint) to completely and thoroughly test the complex software normally seen in DOD weapon systems with embedded computer resources.

The logical solution—manual backup—is usually also impractical, because it often involves retaining and maintaining the very system that the newer, computerized wonder was earmarked to replace. The obvious cost impracticality of this approach is not always balanced with operational and user realities.

MANPOWER AND PERSONNEL

The cultural impact of computer technology can be best observed in terms of the young computer scientists and engineers coming on the scene today, and especially in the future. I have personally observed a number of these individuals whose primary loyalty is to the "state of the art" or the technology itself. The employer or employing organization receives secondary loyalty at best. Perhaps this can be illustrated by cases where programmer turnover has reached 33 percent per year. Couple this with a projected 40 percent programmer shortage by 1990, and it may foretell a significant problem. Since software maintenance is accomplished by means of software modifications, the individual who performs this function must be capable of designing the software in the first place. Hence, the skill level of a software maintainer must be as good as or better than that of the software design engineer.

FIGURE 2
Types of Software



The technology of an existing software program generally stands still, since its basic structure and language are fixed at the outset of design. Hence, the incentive for individuals (whose primary loyalty is to the latest technology) to maintain existing software is certainly lacking. Difficulties in attracting and retaining good people to maintain existing software can be anticipated.

SUPPORT AND TEST EQUIPMENT

As shown in Figure 2, support software consists of computer programs that are used for the development and maintenance of other computer programs. These consist of building tools (for building new programs) and maintenance tools (for maintaining existing programs). The often overlooked fact of support software is not that it exists, but that there is so much of it. One mature aircraft program, for example, has used 1,500 support software programs to build and maintain seven operational flight programs.

System-level troubleshooting techniques for a modern system with ECR include performance monitoring/fault isolation capability (on-line maintenance and diagnostic programs) whose foundation is in software. Hopefully, this will help maintenance people to initially recognize a failure and distinguish (at the sys-

tems level) between a hardware and software failure.

At hardware repair echelons such as intermediate and depot, additional support and test equipment is usually found. Expensive, sophisticated external automatic test equipment (with its associated test program sets) is usually seen in a modern hardware depot facility.

Again, all of these factors point to one thing—significant costs, because all of the operational and support and test software must be maintained.

TRAINING AND TRAINING DEVICES

Training Devices. Computer-aided instruction is a new tool available with great promise. Self-paced instruction is also proving to be an efficient learning tool gaining greater acceptance among the services every day. Both types of instruction, however, usually have some foundation in software. Simulators or trainers that simulate the operational system have been used in the past, and are increasing in sophistication, effectiveness, complexity, and cost. The most modern of these devices are becoming software-intensive systems. Also available are trainers that utilize the operational hardware loaded with a training program in order to function as a training device. All require support.

Training. Operator and maintenance training for systems with embedded computer resources must be provided in a timely manner to support planned introduction rates of these systems. This effort must include instruction in software, depending on the training purpose and operational system itself.

An overall training plan is usually prepared, and includes planned instruction in formal schools and informal "on the job" training. Impact on existing training is also addressed.

Instruction in system operation, organizational-level maintenance, intermediate-level maintenance, and depot-level maintenance is normally provided. Hardware and software are addressed at each level to the degree dictated by the operational and maintenance concepts. New material introduction team training, instructor training, and refresher training must also be addressed.

Technical Data. Technical data has sometimes been called "software," but it is confusing to refer to it in this fashion. Technical data refers to myriad written instructions and records. Many managers attempt to avoid initial acquisition costs by not documenting software and planning to use life-cycle prime contractor support, or buying the data at an unspecified later date. Others may delete documentation requirements by uttering the famous last words, "It'll never fail anyway." Still others may get into trouble by not establishing clear data rights to support software or other less-visible items. The technical data problem is not necessarily solved by delivery or truckloads of program documentation. Careful selection and tailoring of software documentation conventions and techniques is

essential. The currency and accuracy of delivered data is of equal importance, and steps must be taken to ensure strict configuration management of software. Managers must be meticulous in selecting the items of documentation required to support a system with embedded computer resources.

Language is also of keen interest here. English is the common language used for years in the hardware world. The English language vocabulary in the software world, however, has not yet matured into a standard set of words and meanings. The number of different computer programming higher-order languages (HOL) is also considerable—CMS-2, JOVIAL, COBOL, FORTRAN, Ada, ATLAS, etc. The problem is compounded in those situations where different dialects of the same higher-order language are used. Also of concern are those situations where assembly language is used to make fixes to programs written in HOL. Language standardization and control is a unique and significant software problem.

PACKAGING, HANDLING, STORAGE, AND TRANSPORTATION (PHST)

This includes actions and requirements necessary to ensure the capability to package, handle, transport, and store all equipment and items of support. Containers, forklift trucks, cargo aircraft, warehouses, commercial transport, security, packing materials, paperwork, transport schedules, preservation, cargo ships, dockworkers, pipelines, and a host of similar considerations characterize PHST. The key emphasis is in the avoidance of damage or deterioration in the safe and timely movement and storage of the system.

PHST is generally more of a problem for hardware vice software. Hardware is usually large and bulky, whereas software can be easily contained in a single reel of magnetic tape, for example. Hardware damage in transport or handling is usually repaired; software damage is usually attributable to damage to the media conveying the software program or alteration of the state of information in the media, and is "repaired" usually by reissuing or duplicating the program on new media.

Extended storage can pose a problem for volatile computer memories, whose contents may be lost or altered. Hardware can be expected to deteriorate with age—it wears out. Although software itself does not wear out, its media wear out. Software also tends to become obsolete very quickly due to our rapidly advancing software state of the art.

SUPPLY SUPPORT

This involves ensuring that spares (hardware spares, computer programs, and repair parts) required to operate and maintain the system are provided on a timely basis. Consumable/expendable items (such as computer printer paper, batteries, printer ribbons, etc.) are also included here. Software supply support must

provision for and routinely resupply media (printer paper, cards, magnetic and paper tapes, etc.) used to transfer/transport computer programs.

Standardization of computer hardware is often done in an attempt to minimize hardware support problems over the life cycle. Unfortunately, standard computer hardware imposes two significant constraints on the software designer—available memory and speed. Software standardization is making progress, but things have not yet reached the point where national stock numbers are assigned to existing computer programs or computer program components.

FACILITIES

The facility (programming support center) required to develop and produce the software is usually identical to the facility required to maintain the software. These aids comprise the programming support center and can be broken down into the following:

—Software development laboratory
—Hardware integration laboratory
—A test system (for final checkout)

Conclusion

The state of the art in software design technology is advancing at a more rapid rate than the state of the art in software logistics thinking. The artificial cultural distinction between the hardware and the software communities in the logistics world must cease because current technological trends demand it. Every system with embedded computer resources is locking us into sustained levels of effort from very expensive people over the life cycle of these systems. Software changes with hardware impact (and hardware changes with software impact) should have given the first signals. More obvious, however, are the implications of firmware—software logic permanently (or semi-permanently) embedded in hardware. This new area of technology has the characteristics of both hardware and software, and requires a unified hardware/software approach to maintenance.

There simply is no alternative in the future to thoroughly integrating software into logistic support. The technology demands it. But are we, the people, ready to cope with it?

A Management Technique for Software Acquisition

Major Howard W. Wendt, USAF
Michael W. Evans

For the software manager, the pitfalls in software acquisition are numerous. To the uninitiated, the labyrinth of software development is overwhelming. The terminology is unclear, the technology is complex, and acquisition, administration, and management of technical personnel are seemingly impossible.

The manager too often throws up his hands in the midst of the development crises. He fails to recognize that although the form of the software management problem may be unique, the specific management techniques, practices, and procedures required to develop, integrate, and demonstrate software are based on good engineering management principles.

The characteristics of the software product that contribute significantly to the management dilemma are the lack of physical products associated with software development, and the complexity of the management program necessary for control of the software development.

Contracting emphasis for software is being placed on use of a structured development environment by the contractor. It is just as incumbent on the government, the procuring office, and the in-plant representative to structure their management and surveillance approaches to the software contract and its development. This paper discusses an approach to this government responsibility, and presents management tools that can be applied during the pre-award, post-award and production phases of software development. Much of this discussion, however, may be directly affected by the availability of trained personnel.

Lesson Learned from Software Management

Software development is a complex process requiring the careful integration of diverse disciplines, technical activities, and administrative project controls. Too often, software projects are implemented by contractors without first defining and implementing firm plans to project how the project will be developed.

Major Howard W. Wendt, USAF, is Commander, DCASPRO-Ford, Palo Alto, Calif. He previously held positions in NATO Logistics Plans at HQ USAF, and at the AGM Joint Program Office, Arlington, Va. Major Wendt holds a B.A. degree in economics from the University of Nebraska, and an M.S. degree in computer systems administration from George Washington University. He is also a graduate of DSMC's Program Management Course.

Michael W. Evans is Vice President for Management Sciences at Integrated Computer Engineering, Cupertino, Calif. He formerly was Manager of the Standards, Procedures, and Control Department with the WDL Division of Ford Aerospace, and has also held positions with IBM, UNIVAC, and Federal Electric Corporation in the areas of programming management and software test and integration.

The somewhat common, though disastrous, approach to software planning is to manage software in comparative isolation, building on previous problems. A more sensible approach is to complete planning functions of the program at the outset of the project, assuring that elements of each category from project definition to personnel planning are in place from the beginning.

Documentation cost, and the development schedule are the key elements in any software development project. These represent the only visible data products available to customers and software managers for evaluation of project progress and integrity. Unfortunately, accurate derivation of these schedules is difficult; monitoring of progress and cost against scheduled milestones is not achieved. Evaluation of the effectiveness of the development process is even more difficult, considering the complexity of software management problem. Still, management controls must be defined and implemented to assure schedule and budget compliance, software quality, and correspondence to user requirements.

The dilemma facing government project personnel is determining how to measure project progress and cost trends in a timely and effective manner, considering the myriad technical, administrative, and project data that must be reviewed and evaluated. The contractor looks on this interface as pacing the development, often claiming contractual impact or change in scope if there are delays associated with conduct of document or development reviews or audits. Comments or suggestions resulting from the reviews, audits, or project-progress reviews may be treated as contractual direction, even if they are couched in informal terms, greatly limiting options available to the reviewer.

The only alternative to the government project manager when assessing or evaluating project cost and schedule is to tailor review activities to a predefined structure consistent with the technical administrative environment of the project. This paper discusses the products and documentation of the development environment that should be included in a project review, and how this environment can be managed.

Planning Concepts

Development of software must be controlled through monitoring development of deliverable and non-deliverable internal planning documents and procedures. The plans and procedures should be subject to project and government review and approval.

The technical and project-control-structures approach discussed here provides the tools necessary for a smooth project flow throughout the development period. These tools allow the government to identify the management structure, technical and project controls, and the test and integration methodology, all of which must be defined and coordinated at the outset of the program.

It is not enough to simply contract for structured development of software. The government must plan and implement a surveillance environment that will ensure visibility of the contractor's program. This surveillance environment must include the basic management elements for software development and a structured cost monitoring program. The cost monitoring program will vary from project to project. In some cases, C/SCSC or C/SSR cost systems will be contracted for. In other cases, the government will be utilizing internal, contractor-maintained systems.

The surveillance environment must be in place from pre-award to final delivery. This government surveillance at the beginning of the program is even more important in software than in hardware. In software, the contractor is being asked to do extensive planning at program inception. It is just as important for the government to have procedures and tools in place to review these plans, and ensure they are implemented in a timely manner.

The following sections present an approach for government surveillance of the contractor's documentation, planning, and implementation during software development, which must be used in conjunction with the established cost monitoring program. The development topics discussed below must be considered and weighed during all subsequent cost and schedule reviews. These elements are discussed briefly in the next section. The important principle in this approach, however, is not the specific elements designated for review, but that government surveillance be preplanned, structured to fit the program, put in place early, and used.

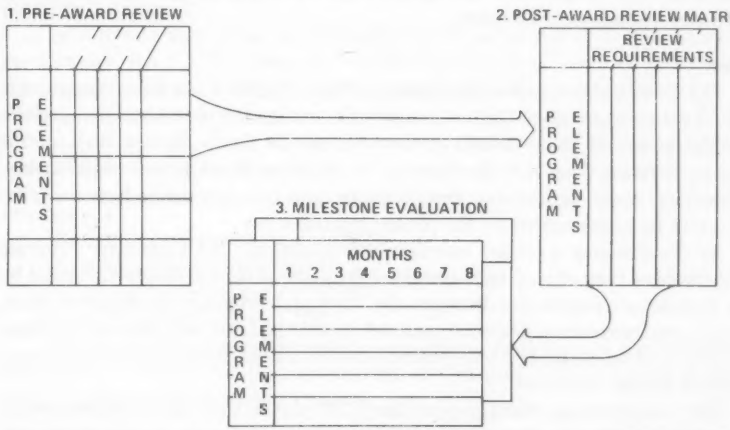
Surveillance Overview

An effective approach toward satisfying the proposal evaluation requirements, and the equally essential function of providing a basis for later project evaluation, is illustrated in Figure 1. Using this approach, the evaluator defines an ordered set of matrices and subsequent surveillance charts to describe the software development environment as identified in the contractor's proposal documentation.

The matrices provide the evaluator a standard set of project criteria for evaluating the completeness of the documentation and the integrity of the proposed planning. Each program element defined in the evaluation matrix identifies, at a high level, the minimum components of the development environment necessary to satisfy customers' needs.

The matrix is used initially during the pre-award time frame to evaluate the proposal and to establish the project environment based on proposed MIL-STDs. Upon conclusion of this evaluation, not only is a judgment developed on the

FIGURE 1
Overview Matrix



worth of the proposed approach, but the development structure has been defined at the top level for later evaluation.

The second matrix utilizes the same elements as the first, but now defines surveillance responsibilities of the customer, the in-plant representative, and the contractor.

The result is a concise matrix of customer priorities where assistance is needed from the in-plant representative, and the expected surveillance and audit responsibilities of the contractor.

The final elements in the illustration are surveillance milestone charts, discussed in detail later. They are a summation of the management structure defined in the matrices. These are used logically to plan and monitor the software management activity relative to the existing program development milestones, i.e., SRR, SDR, PDR, CDR, and the design/code/testing of the various computer program configuration items (CPCIs).

Management Elements

For the purpose of this matrix management approach, software management has to be divided into eight sections: plans, baseline, control, reviews, configuration management, change control actions, change control documentation, test,

and audit. These categories were selected for their management visibility needed for surveillance, not for the technical insight into physically developing the software. The subcategories, described briefly below, are only representative of software management elements.

PLANS

The Computer Program Development Plan (CPDP) is the basic documentation of the software organization and control structure. It provides a comprehensive definition of basic criteria to be followed in the implementation of the project, defining what is to be done by the software development organization. Subsequent plans, procedures, specifications, and manuals define how the criteria are to be implemented by the project organization.

As the primary software management document, the Computer Program Development Plan should be maintained throughout the development period by the contractor. Correlation between the computer program development plan, project implementation practices, and deliverable and non-deliverable documentation should be assured by independent audits conducted by contractor and government project personnel.

The Configuration Management Plan (CM) should describe the techniques to be used in performing the configuration-management function, and should discuss the objective of configuration management, and the importance of baselines. The four basic areas—configuration identification, configuration control, configuration status accounting, and configuration audits—should be discussed, as well as who will be responsible for developing and implementing the procedures. The functions of the Program Support Library (PSL) should be discussed, as well as what will be contained in the PSL. Project standards, the means of controlling PSL releases and changes to materials in the PSL, should be included as well as who approves releases and changes. The functions of the PSL librarian, baselines, specifications, support documents, and internal documents, should be addressed. The plan should also discuss discrepancy reporting procedures, and change control documents, and the function and membership of configuration control boards should be described.

The Quality Assurance Plan (QA) will usually be an in-house document, referenced, and briefly discussed in the CPDP. The plan should cover configuration identification, configuration control, status accounting, audits, tests, corrective actions, library controls, etc. The scope and delivery status, and use of the plan should be defined early—at pre-award. Audits and reviews must then ensure the plan is being followed.

The Programmers' Notebook is an internal volume outlining the software development clerical structure for designers and coders. This ensures that the

actual design and code will be developed uniformly. Other practices necessary to produce consistent documents are set forth in this notebook.

BASELINE CONTROL

The software specifications normally will be called out as a contract document review list (CDRL) item, with some level of detail given to content and format requirements. During pre-award, any specifications not called out in the CDRL, or intentions of either the customer or contractor to expand on the CDRL, should be discussed. Later, during post-award, the details of timing and review responsibilities must be determined.

REVIEWS

—Systems requirement review (SRR)
—Systems design review (SDR)
—Preliminary design review (PDR)
—Critical design review (CDR)

These are the classical program management reviews and are included in this matrix to ensure that their application and purpose in the software environment is understood by all participants.

CONFIGURATION MANAGEMENT

Librarian Tasking: The use of a strong librarian for software configuration management is an essential control element. The librarian role or comparable structure should be discussed and defined at the outset of the program.

Software Change Review Board Meetings: Some programs will have a separate software review board; others will incorporate software actions into a central program Configuration Change Board (CCB). Whatever the structure, change-action management is a critical element in software configuration control. Subsequent board minutes and change actions should be published, available, and reviewed. Discussion of their status is essential.

Unit Development Folder (UDF): This is the central location where all information concerning the modules making up each unit shall be located. This information includes the design, test, and status information for modules that have been grouped into units for administration and management control. The format, content, and control of the UDF should be specified in the Programmer's Notebook.

CHANGE CONTROL ACTIONS

While Class I changes are quite structured, Class II changes, internal changes, and waivers are managed in many ways. While various practices are acceptable, agreement should be reached on the specifics of the procedures and how they will be implemented.

CHANGE CONTROL DOCUMENTATION

Specification change notices, configuration indexes, and version-description documents should be identified and structured by the contract via the statement of work (SOW) or CDRLs. Their intended format, use, and availability must be defined.

TEST

Test items on the matrix are fairly standard and should be easily defined. The availability and format of internal documents and test results are not so straightforward and should be established early. Internal audits and walkthroughs will probably be conducted, but how well they are documented, who will have review authority, and who will participate must be defined.

AUDIT

The formal FCA and PCA audits should be well documented. Testing and participation is worth documenting at the outset of the program.

Pre-Award Planning

During the pre-award time frame, the list of elements provides an agenda for review of a contractor's proposal (Figure 2). For example, comparison and discussion of proposed elements relative to the list can help define what a reference means by "configuration management," i.e., will there be a designated librarian, a software review board, or an automated library? The contractor will also be able to evaluate the level of participation and control planned by the government plant office and the customer as to what internal documents will be reviewed and audited. This top-level matrix is then documented by specific planning information provided in the Computer Program Development Plan analysis.

The MIL-STDs reference at the top of the matrix provide a basic understanding of standard program structure. During the pre-award phase, the MIL-STD called for in the Request for Proposal (RFP) should be reviewed for application in relation to the listed program elements. A key point to review with the contractor is the extent of compliance anticipated for each MIL-STD. There may be a difference in the interpretation of the coverage required by the MIL-STD, tailoring of which should always be a consideration. If the contractor is bidding based on a

FIGURE 2
Pre-Award Review

PROGRAM ELEMENT		MIL-S-52779	MIL-STD-1679	MIL-STD-480	MIL-STD-483	MIL-STD-490	MIL-STD-1521A	MIL-D-9858	OTHER	COMMENTS
A. PLANS	CPDP CM PLAN QA PLAN PROGRAMMER'S NOTEBOOK									
B. BASELINE CONTROL	REQUIREMENTS SPEC (A) DEVELOPMENT SPEC (B_5/PT I) PRODUCTION SPEC (C_5/PT II)									
C. REVIEWS	SRR SDR PDR CDR									
D. CONFIGURATION MANAGEMENT	LIBRARIAN TASKING SOFTWARE CHANGE REVIEW BOARD MTGS. - minutes - action items FILE STRUCTURE UDF									
E. CHANGE CONTROL ACTIONS	CLASS I CHANGES CLASS II CHANGES REQUEST FOR WAIVER INFORMAL CHANGES									
F. CHANGE CONTROL DOCUMENTATION	SPEC CHANGE NOTICES CONFIGURATION INDEXES VERSION DESCRIPTION DOCUMENTS									
G. TEST	PROCEDURES COMPUTER PROGRAM TEST AND EVAL. - unit test - (test walkthrough) - I & V test - (test data) PRELIMINARY QUAL. TEST (PQT) FORMAL QUAL. TEST (FQT)									
H. AUDIT	FUNCTIONAL CONFIGURATION AUDIT (FCA) PHYSICAL CONFIGURATION AUDIT (PCA)									

tailored approach to any of the MIL-STD, this should be defined. By the same token, the contractor should be briefed on the government interpretation of each element of the MIL-STD, and the degree of compliance expected.

The result of this review is not necessarily to dictate how the program should be structured, but to mutually understand what the proposed structure is, concur on the probable adequacy of its use, and to document the structure as a baseline for later program surveillance.

FIGURE 3
Post-Award Review

PROGRAM ELEMENT		DD 1423	CUSTOMER APPROVAL	CUSTOMER REVIEW	PLANT REP OFFICE APPROVAL	PLANT REP OFFICE REVIEW	CONTRACTOR QA APPROVAL	CONTRACTOR QA REVIEW		COMMENTS
A. PLANS	CPDP CM PLAN QA PLAN PROGRAMMER'S NOTEBOOK									
B. BASELINE CONTROL	REQUIREMENTS SPEC (A) DEVELOPMENT SPEC (B_5/PT I) PRODUCTION SPEC (C_5/PT II)									
C. REVIEWS	SRR SDR PDR CDR									
D. CONFIGURATION MANAGEMENT	LIBRARIAN TASKING SOFTWARE CHANGE REVIEW BOARD MTGS – MINUTES – ACTION ITEMS FILE STRUCTURE UDF									
E. CHANGE CONTROL ACTIONS	CLASS I CHANGES CLASS II CHANGES REQUEST FOR WAIVER INFORMAL CHANGES									
F. CHANGE CONTROL DOCUMENTATION	SPEC CHANGE NOTICES CONFIGURATION INDEXES VERSION DESCRIPTION DOCUMENTS									
G. TEST	PROCEDURES COMPUTER PROGRAM TEST AND EVAL. – UNIT TEST – (TEST WALKTHROUGH) – I & V TEST – (TEST DATA) PRELIMINARY QUAL. TEST (PQT) FORMAL QUAL. TEST (FQT)									
H. AUDIT	FUNCTIONAL CONFIGURATION AUDIT (FCA) PHYSICAL CONFIGURATION AUDIT (PCA)									

Post-Award Planning

The post-award time frame is a critical period in the software-development cycle. At this phase, the elements of the matrix, which were previously related to MIL-STDs and "what" was to be done, can be reviewed relative to "who" will be responsible for the audits and reviews of each element. (Figure 3). The matrix elements now define the key management items in the particular software project. At this time there may be further tailoring of specific elements; i.e., more contractor audits and reviews may have been contracted for, more documentation may be required, or management tasks may have been tailored. The post-award review is the best time to update these elements, if necessary, and ensure everyone mutually agrees on the scope of each.

The contractor now has a good indication of the software management elements he can expect the government to review. Too often, a contractor bids on a program and accepts all the MIL-STDs called out in the RFP. But he does not really anticipate full compliance during program development, possibly because that customer never asked for them on past contracts, or the contractor may not understand the full impact of a particular MIL-STD requirement.

As each element is discussed, it is important to agree who has the primary interest in conducting reviews. The customer has an obvious responsibility to review and approve items called out as a Contract Documentation Review List (CDRL) item. The plant representative office may or may not have approval authority on a CDRL item, depending on how the CDRL is written. These relationships need to be highlighted and discussed. This plant representative/customer review of CDRLs usually brings up many issues that should be addressed. Typically, questions arise on such document distribution as what documents must the plant representative have to do the agreed job, and what documents can be reviewed early as in-plant drafts. Additionally, the customer may have a special interest in, or concern for, a non-deliverable item, and may want to personally review the items during each plant visit. These special interests should be noted on the matrix, and a balance agreed to that will ensure these areas are not over-emphasized by duplicate reviews while others are overlooked completely.

The matrix approach is also useful in subcontract management. From the in-plant representative point of view, the completed matrix is a documented presentation of the management expectations of the customer, and the program agreements of the prime contractor. Exceptions taken by a subcontractor would need to be specifically documented in the flow-down contract to him. Use of this matrix review with a subcontractor should bring these exceptions to light quickly, and discussions should begin on program impact. Special attention needs to be given during pre-award reviews to subcontract management during pre-award reviews to ensure that the proper MIL-STD and MIL-Q requirements will flow-down to the subcontractors, thus facilitating the post-award management discussions.

Program Surveillance

At this point, the key management elements are visible, defined, and scoped. Review responsibilities and special review requirements have been highlighted. The final level of review is the milestone tracking format (Figures 4 and 5). The purpose now is to document the review requirements relative to the actual software development milestones as established by the contractor's program management plans.

Some dates can be taken directly from existing planning documents. Reviews such as SRR, SDR, PDR, and CRS have been scheduled; anticipated delivery dates for "A", B_s, and C_s specifications should be available. Some schedule data, however, are not so obvious.

The first chart (Figure 4) is useful primarily as a tool during the early program development. The items included will generally refer to establishing the initial management environment, while the second page begins the actual tracking of the software development and ongoing reviews of the management environment.

Figure 4, Item 1, "Contract Reviews," reflects the initial contracting review meeting, which has been discussed in the previous matrices.

Item 2, "Contractor Plans Review," begins the review of the contracted and in-house documentation. These dates include documentation delivery. Some documents will be delivered against a CDRL and delivery status will be tracked contractually. Documentation problems arise with the in-house documents, i.e., configuration management plans, quality assurance plans, test plans, design, code and test procedures, which are necessary for program management, but always seem to be "almost ready" or "in draft" format. It is difficult to measure the adequacy of the software management when the management documents have not been formally presented for review.

Tracking and ensuring timely delivery of these management documents becomes an important function of this milestone chart in the early phases of the program. The idea is not necessarily to dictate when delivery will be, but to mutually agree on a reasonable future date, and then track delivery or slips to that date. A missed delivery date on design procedures is not critical, but a series of slips begins to suggest poor planning, poor management, or program technical problems.

As items are tracked and reviews completed, the milestone chart becomes a common baseline for monthly communication between the customer and the in-plant representative office.

Item 7, "Resource Allocation," again integrates the cost and schedule aspects into the software elements presented in this paper. During this subsequent program surveillance, the cost and schedule audits must be planned to address the

FIGURE 4
Initial Acquisition Milestones

Program Elements	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1.Contract Review												
PCO/ACO Meetings												
Preaward/Post Award Meeting												
MIL-STD/Spec Review												
MOU/Surveillance Matrix												
2.Contractor Plans Review												
CPDP												
Q.A. Plan												
C.M. Plan												
Test Plan												
3.Program Reviews												
Development Spec Audit												
Requirements Spec Audit												
Production Spec Audit												
4.Librarian Tasking/Procedures Review												
5.Q.A. Tasking/Procedures Review												
6.SCRB Procedures Review												
Resource Allocation												
Cost Accounts/Schedule Audit												
W.B.S. Plan/Implementation Audit												
8.GFE/GFP Accountability Audit												
9.Contractor Program Reviews												
Software Review												
10.Customer Program Reviews												
Software Review												
11.Plant Rep Reviews												

FIGURE 5
Program Surveillance Milestones

Program Elements		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1. Program Reviews:	SRR												
	PDR												
	CDR												
2. CPCI Status:	CPCI No. 1												
	CPCI No. 2												
	CPCI No. 3												
	CPCI No. 4												
	CPCI No. 5												
	CPCI No. 6												
	CPCI No. 7												
3. Integration Test:	Build No. 1												
	Build No. 2												
	Build No. 3												
4. H.W./S.W. Integration													
5. Contractor Audits													
6. Contractor Procedures Review													
7. Plant Rep Procedures Review													
8. Plan Rep Procedures Audit													

software development cycle. As the development environment is defined and understood, the specific reviews of the WBS and cost accounts can be managed, based on actual program knowledge and experience.

The balance of the items in Figure 4 are reviews, either the review of a functional area, 4-8, or the various informal reviews and meetings, 9-11, which will be continuous through the life of the development process.

The statusing of the regular reviews and meetings is a significant element for customer and plant-representative management. Neither office will usually have sufficient personnel to keep up with everything. Reviews need to be coordinated to eliminate duplication and to ensure the critical areas are covered. Where the contractor is reviewing or auditing certain activities, the plant representative should perform periodic evaluations of selected contractor activities to assure the effectiveness of the contractor's QA effort.

The second milestone page begins with a baseline listing of reviews: SRR, PDR, and CDR. Status monitoring of the individual CPCIs is the important element. As each CPCI is tracked, the anticipated PDR, CDR, and test dates are noted above the bar graph. As a date is missed the traceability is maintained and, if a negative trend is developing, this approach should highlight it.

Again, this information is available from the contractor but, in a software environment, the essential elements that are to be reviewed must be highlighted.

Once the basic CPCI development and integration schedules are highlighted, planning can begin on reviews and audits. The reviews planning, as discussed earlier, are also important from a manpower viewpoint. Not only can a duplicate review be a waste of the reviewer's time, but will unnecessarily tie up time of the contractor. Further consideration must also be given to the time sequence of the reviews. Some reviews should be completed prior to PDR; others on a reoccurring basis. Manpower and available time again become a key consideration in arranging the sequence of the reviews and integrating them together.

Conclusion

Evaluation of contractors' software development planning, as expressed in their proposals and planning documentation, is a difficult process considering the inherent complexity of effective software planning specifications, and the diverse nature of the tasks required to successfully develop, document, integrate, and demonstrate a software product. There is an inherent difficulty in objectively comparing several diverse approaches to the same problem and selecting the most effective.

There are many additional approaches possible for a more in-depth evaluation, ranging from simple comparative readings and evaluation of documents contents, through a section-by-section statistical ranking of the proposed management elements based on a predefined development standard.

While program tailoring and structured software development is a recognized practice for most contractors, the procuring activities and plant-representative office must also recognize their responsibilities to preplan their approach to software management, and to ensure that the contractor understands the who, what, where, and how of that approach.||

The Evolving Nature of the C^3 Systems Acquisition Process

Robert B. Doane

Problems in command, control, and communications (C^3) systems acquisition management are not new. The problems associated with the proliferation of Air Force command, control, and communication systems in the 1950s led to the consolidation of Air Force C^3 systems development and acquisition at Hanscom Air Force Base, Mass., in 1961 in what is now the Electronic Systems Division, Air Force Systems Command.

During the ensuing years, the Air Force, as well as the other services, have been less than satisfied with the results of efforts to field, within a reasonable period of time, operationally effective C^3 systems. The principal reasons for the poor record of C^3 systems acquisition in the 1960s were, in our view (1) the lack of ability on the part of the operational command to articulate system needs consistent with the rapidly evolving potential provided by the computer explosion; and (2) unforeseen weaknesses in the classic weapon system acquisition process when it was applied to C^3 acquisition.

The Impact of Computer Technology

In the late 1950s and early '60s, the rapid introduction of computer technology, with its promises and problems, promulgated a new revolution within the military structure. Specifically, the computer offered the potential to extend the human mind and, in a more subtle sense, the human nervous system. This concept is not yet fully understood in terms of its military potential, nor has it yielded to an easy description in terms of user requirements and system design specifications. While the nervous system can be described in terms of physical attributes (neurons, synapses) and function (thinking, reacting, controlling), certain crucial characteristics cannot be described. For instance, given that you can think—what do you think about, to what level of detail, and under what conditions? When do you react? How do you react? When a sense (sight for example) is lost, how do you compensate?

While computer technology offers a revolutionary potential, it is plagued by problems. The industry, while very large, is in its infancy and lacks maturity stability, and professional standards and controls.

At the same time, computer hardware technology is burgeoning, having produced several generations of equipment in 25 years, each an order of magnitude

---

Robert B. Doane is the Deputy for Technical Operations and Product Assurance at the Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Mass. At ESD, he also has served as Director and Assistant Deputy for Engineering and Technology, Assistant Deputy for Command Systems, Assistant Deputy for Planning and Technology, and Deputy for Command and Management Systems. Mr. Doane holds a B.S. degree and master's degrees in aeronautical weapon systems and management from Massachusetts Institute of Technology.

more powerful than its predecessor. This has led to applications that are hardware driven, and to optimization of systems based on hardware costs, in spite of the painfully accrued experience that software is the predominant cost driver, now approaching 80 percent of the cost of a typical system. Even more important is the point that software is the very substance that makes a C^3 system a "genius or an idiot." On a conceptual level, the technology is characterized by myths, mystique, and fear, and a belief that computers come off the assembly line with an innate intelligence built in as if by some magic, ready to respond to the will of the user.

The Air Force began to apply computer technology to C^3 systems in the late 1950s and early 1960s. In acquiring the first generation of C^3 systems, we naturally, but unfortunately, assumed that prior weapons systems acquisition strategies and experience were appropriate, and followed their precepts. The acquisition efforts of the '60s can be characterized as follows:

—We formed large system program offices (SPOs) tailored to the ballistic missile acquisition experience, and followed the full cycle of design, development, acquisition, and deployment.

—Development was generally away from the user, with little user involvement.

—We attempted long development cycles against operational dates 5 or more years into the future.

—We contracted with industry for "applications" software that unknowingly represented the personality and operational "tactics" of the user.

—We assumed that system requirements were stable and not prone to change.

—We attempted massive increases in capability off-line to the user.

—We used sophisticated system management procedures (AFR 375/800 series) that had proved successful for weapons systems acquisitions that, in general, enjoyed prototyping, followed by large production runs. By contrast, C^3 systems were generally one-of-a-kind.

During the '60s, we began developing the first generation of information or management systems, the acquisition of which took a slightly different tack than that used in weapons-oriented C^3 systems acquisition.

—We treated management systems as machine aids to the bookkeeper.

—We used no SPOs—systems management was distributed, and generally ill-defined.

—Attention was focused on hardware available from the commercial market with little, if any, consideration given to application software.

—By default, the users designed and developed application software in-house, with the result that development was evolutionary—build a little, use a little, build some more.

—The systems were relatively flexible to changing requirements. Because the user was his own architect, he could weigh new requirements against available development resources and make the necessary trade-offs.
—Equipment management procedures were developed that were essentially silent on the subject of software (AFR 300 series).

In tracing these two types of computer or software-dependent systems into the late '60s and early '70s, we observed that the distinction between C³ systems on the one hand, and management and information systems on the other, was artificial. The technologists complemented this observation by producing sophisticated, large, general-purpose computers, which made it possible to support many functional areas with one integrated system. This expanded computer power gave rise to the first total information systems, e.g., MACIMS and WWMCCS. However, both regulatory approaches stayed on the scene to provide two different acquisition processes, each with benefits and each with faults.

A few specific acquisition examples will set the stage for what we have learned about the acquisition of computer- or software-dependent systems.

SAC C² SYSTEM (465L)

The acquisition was a classic AFR 375 approach, which exactly fit the characteristics of the C³ system acquisition process described above. The system was developed off-line to the user. The software fell behind in its ability to satisfy the users' requirements. Schedules stretched out, and costs rose to the unacceptable level. The vicious cycle was stopped when the Strategic Air Command (SAC) accepted the system in 1965 and reaccomplished the software in-house.

AIR FORCE INTEGRATED C² SYSTEM (473L)

The classic AFR 375 approach was followed; however, the development was closely coupled to the user. Because the user, HQ USAF, was the directing and funding authority, requirements were vague at the outset and frequently changed by decree, with no attendent change in program scope, schedule, or funding. The program soon became uncontrollable. The result was a large contract overrun and program cancellation in 1967. Program difficulties were compounded by attempting simultaneous computer hardware and software development—a lesson high on our list of lessons learned!

ADVANCED LOGISTIC SYSTEM (ALS)

The ALS was an AFR 300 series acquisition, with no SPO or development command involvement. The computer-buying agency delivered $83 million worth of computer parts to the loading docks at the Air Force Logistics Command, and drove away. Ambitious requirements, no system acquisition manage-

ment structure, focus on hardware, and other factors led to one of the Air Force's most widely publicized acquisition failures. Needless to say, the program was cancelled, and new lessons were learned!

Nature of C³ Systems

Unfortuntely, problems with the acquisition of software-dependent C³ systems do not stop with these three examples. After 20 years of experience we are still reaching for the key that will unlock the door to a more effective C³ systems acquisition process. The elusive search for a more viable software-dependent-systems acquisition process requires that we view the problems associated with the past failures as mere symptoms of a more fundamental problem. With this in mind, let us be introspective and compare the characteristics of C³ systems with those of weapons systems, for it is in these comparisons that the insight required to establish a more effective process for software-dependent C³ system acquisition lies. (Figure 1)

Communications, command, and control system requirements are intrinsically evolutionary, partly because they must operate in a constantly, but not always predictably, changing environment, and because they must support human decision-making, a process that cannot be completely specified *a priori*. This aspect of C³ systems implies a continuous and close cooperation between the end users of the system and the developers of the system, and the need for flexibility in the definition of both system requirements and end-item capabilities. The prevailing Air Force development and acquisition practice that treats C³ systems as if they were stable rather than changing, or akin to mass-produced items rather than one-of-a-kind, has been out of step with this reality. This situation has been further aggravated by Air Force practices that do not foster continuous, close cooperation between end users and developers of C³ systems.

The appropriateness of the basic life-cycle concept that builds on requirements and specifications is being increasingly questioned, particularly as it pertains to rapidly changing, computer-intensive C³ systems. Once we agree that, even collectively, we are incapable of accurately stating what is needed (or will be needed), because the requirement itself is subject to constant change and redefinition on a time scale much shorter than the time required to develop the system, then the development process itself must be viewed as dynamic, and not amendable to an invariant sequence of activities.

Organizational factors add another layer of unique characteristics to C³ acquisition. Most C³ systems cut across service lines and interface at many points both vertically and laterally. Key users may be service unit commanders, CINCs, or the National Command Authority. Systems typically must be interoperable with

FIGURE 1
C³ System - Weapon System Paradox

AREA	C³ SYSTEMS	WEAPONS SYSTEMS
Multiple Units	Mind Extender	Strength Extender
Revolutionary	Personality	Impersonal
Prod/Dollar Intensive	Complex & Perishable	Discrete & Retainable
High	Multi-Purpose	Single Purpose
Process Control	One-Of-A-Kind	Production Quantities
	Interactive, Internetted & Interdependent	Stand Alone
	One System Serves All	
	Evolutionary	
	Labor/Coord Intensive	
	Low	
	Dynamic Interactive	

many other systems designed at different times with different emphasis. Researchers, designers, and users are likely to be in different organizations and locations. We must recognize these facts of life, and assure that our acquisition practices and procedures accommodate these realities.

Communication, command, and control systems are generally defined as the complete collection of sensors, facilities, equipment, computers, software, communications, procedures, and personnel essential to the commander for the planning, directing, and controlling the operations of assigned forces pursuant to the mission assigned. From a technology point of view, C³ needs are met by building on computer science, systems and information science, and communications. The systems must truly deal with distributed commodities, resources, and services on a time-critical basis. Current systems were frequently designed on the basis of insufficient data, and often lack the flexibility to grow gracefully. The computer hardware capabilities have been doubling every 2 years or so for the past 20 years with no end in sight. The supporting software systems are changing just as fast in providing additional baseline capabilities which, when combined with the current lengthy, rigid acquisition process, makes obsolete (relative to the rapidly moving new technology) any command and control system before the design has even been completed.

The issue is how to take advantage of technology and evolve computer- and software-dependent systems, as opposed to the usual weapon system hardware-based system whose requirements and specifications are capable of being

parameterized and quantified in advance.

Unlike the typical weapon system, it is difficult, and sometimes impossible, to develop a firm operational description of what a C^3 system should do, or a boundary for its users. The JSC Pub 1 definition of command and control is as follows:

> The exercise of authority and direction by properly designated commanders over assigned forces in the accomplishment of his mission. Command and control functions are performed through an arrangement of equipment, communication, facilities and procedures in planning, directing and controlling forces and operations in the accomplishment of his mission.

Under this definition, a C^3 system can be bounded in terms of a level-of-command perspective: A C^3 system for CINCNORAD must support the information needs of the JCS and, in turn, the President. In this context, each aggregation of hardware, software, personnel, and procedures classically described as a C^3 system is, in a requirements sense, a "subsystem" of one overall C^3 system. All of those subsystems are, to some degree, interactive. For acquisition of all these subsystems, centralized acquisition management is not necessary. What is necessary is a concerted effort to define, with a degree of stability, the top-level information needs for all levels of command. Given this, it would be possible to define a distributed system of C^3 elements that can satisfy the hierarchy of C^3 from the "local" (battle) commanders up through the JCS—a very difficult task. Therefore, what we need is an overall C^3 systems architecture and a "system integrator."

The one-of-a-kind aspect of many C^3 systems has resulted in an approach where we attempt to build a complete system, on the first attempt, from paper analyses, rather than relying on experience gained from prototype-systems demonstrations. In another context, the weapons system acquisition philosophy provides for a less-than-complete systems approach in the sense that a limited investment is made to prototype a weapons system and measure its operational effectiveness prior to committing to production. Thus, for weapons systems, the systems acquisition process and philosophy "accepts failure." Current weapons system acquisition practice, when translated into a C^3 acquisition philosophy, has not allowed for limited investment failures; failures are not discovered until a "full" investment is made in the system, because we are producing a one-of-a-kind system. Prototyping has not been deemed necessary, previously, primarily because the cost of prototyping is judged to be too large a portion of an all-up system acquisition investment. It is clear from our experiences that lack of prototyping compounds the problems associated with the acquisition of C^3 systems.

Most important, the personnel and procedural aspects of a C³ system require complete integration of the human element into system design criteria. Consideration of structural, procedural, and personnel impacts must be factored into the development of a C³ system from one day to the next. Unfortunately, most C³ systems have been developed off-line to the operational user so these factors cannot be injected into the system criteria, nor adequately evaluated. Further, no amount of testing in a simulated environment will overcome a user's fear that he is being handed a "pig-in-a-poke" when it is time to cut over to a new system. This feeling is magnified by a natural human resistance to organizational and procedural change. Oddly enough, this user resistance manifests itself in the form of a continuum of requirements changes that preclude delivery of anything less than a perfect system.

The dynamic nature of C³ software is a prime driver of C³ acquisition problems. Currently, we acquire software as a static entity—a direct adaptation of our hardware acquisition procedures. This methodology cannot cope with the need to deal with changing requirements, and stifles industry creativity and productivity necessary to respond to such a need in a competitive, price-driven source selection.

Lastly, we tend to judge the value of a C³ system against the development dollars expended to develop the system. The true value of C³ systems lies in their force-multiplication potential. That is, C³ systems provide potential cost avoidance: Good C³ systems preclude the need for vast numbers of weapons systems by allowing more efficient and intelligent use of the weapons systems in response to a given threat. Unfortunately, no reliable means of quantifying this force-multiplication factor exists.

This introspection into the nature of C³ systems, and the process we have used to acquire C³ capabilities, leads us to conclude that we need to better appreciate the fact that our systems acquisition process must be adapted to recognize the fundamental differences between weapon systems acquisition and C³ systems acquisition. In addition, we must recognize the fact that C³ systems acquisition is greatly impacted and, to some extent, confused by two competitive, yet different, regulatory acquisition management processes (AFR 300 and AFR 800 procedures). A review of the lessons we have learned over the last decade of C³ acquisition can give us the necessary perspective to adequately deal with these two fundamental issues that cloud the C³ systems acquisition business.

Lessons Learned

The following is a sampling of what the Electronic Systems Division has learned as the Air Force's lead division for developing C³ systems:

—The classical SPO management philosophy does not fit well with software-dependent, decision-oriented systems acquisition. Development must be on-line in the sense that the user must be afforded the opportunity to "see, feel, touch, and taste" the system in an operational environment while the system is under development. The user must be his own architect to inject "personality" into the system, and to have the system software truly represent the most current tactics for system development.

—Requirements must be detailed and ambiguous when time comes to commit to implementation. For example, a means of specifying such concepts as graceful degradation must be found, if we are to be able to develop successful systems.

—Requirements for C^3 systems, and C^3 systems acquisition are not understood by senior management. We have not been able to bound the definition of C^3 systems in such a way that senior managers can inject the many nuances of command into the system until the developer is committed to, and well into, systems development.

—The overall military information needs, under various wartime and peacetime scenarios, must be developed to allow for development of individual C^3 systems that are compatible at all levels of command.

—Development and implementation must be incremental. Quantum increases in capability must be avoided. Incremental transfer of the system to the user must be accepted as a fact of life.

—The system must be flexible once designed. Arbitrary hardware restriction (a result of optimizing on hardware costs) must be avoided to allow for effective software evolution.

—The C^3 systems must be designed in recognition of the personality of the man-machine interface, and must recognize and allow for the personality of each new commander. Mechanical replacement of humans (especially commanders!) will fail.

—Prototypes are an absolute necessity. The developer/user team needs operational prototypes to test concepts and development test facilities to allow development of the system in an operational environment without conflicting with operations; just as important is to try out new structural and procedural approaches in concert with the system development. While seemingly increasing the cost of acquisition, this is undoubtedly the key step to reducing our overall C^3 acquisition investment over the long haul. Prototypes do not have to be full-scale mockups of the desired system. In some cases, we could capitalize on the requirement for backup systems for use as developmental prototypes.

—We must optimize on total system cost. Ignoring software costs and restricting programmer productivity through arbitrary hardware restrictions is sheer folly.

The Brooks Bill is often blamed for our optimizing on hardware. This is a simplistic and erroneous argument. The concept of "sizing" hardware to a minimum requirement is ingrained in our weapon systems acquisition heritage. When 80 percent of a systems cost is attributed to software, we can no longer let the 20 percent hardware cost drive the architectural constraints of the system design. To achieve true total-system optimization, we must better understand our overall national information needs. Without this understanding, optimization will tend to be focused on individual user needs, which will not eliminate unnecessary and unknowing duplication of systems among the services.

—Software development is people-dependent and subject to imperfection. Scheduling and cost predictions must reflect this fact. Further, software acquisition management is based on the presumption that rigorous "paper" description of the design and successfully plodding through design reviews will automatically result in a working system, on cost, and on schedule. This is not true.

—Interoperability of systems is a key issue. It cannot be addressed as an afterthought. A vigorous attack on ways of achieving interoperability is essential. Again, a top-down inquiry into our national information needs would be an excellent first step toward achieving interoperability, as well as providing the potential to identify both existing shortfalls and redundancies in our C³ systems.

—We tend to manage by classic weapon system acquisition management criteria. We use the classic parameters of cost, schedule, and technical performance. These parameters must take on a different perspective in an iteratively built, open-ended system acquisition where requirements and system capabilities evolve over time. Better insight into the underlying problems of C³ acquisition is needed to establish a set of management criteria, which, if properly used, will prevent the occurrence of the symptoms of cost overruns, schedule tips, and poor system performance.

—Configuration control of software is a development necessity. However, control must be restricted to the functional requirements of the system and the control of the product after it is built. It is a mistake to make efforts to baseline and rigidly control the design of the product as the design is being developed (through iteration within the analyze, design, code and test process). This calls for rework of AFR 800-14, along with the supporting MIL-STDs and data item descriptions. (There are inconsistencies in these documents with respect to the definition of an allocated baseline, which results in our controlling a design very rigidly, while the design is being created.) Control is absolutely essential, but at the right time.

—The hardware replacement cycle must be lengthened by buying large computers, oversized for the anticipated software development effort. This will reduce the occurrences of software conversion efforts and enhance programmer productivity.

—We try to develop ideal systems through paper analyses. These C^3 systems are conceptualized, validated, developed, and deployed through people-intensive, creative efforts. Mistakes will be made. The inherent goal of acquisition should be ideal systems, but the operational goals should be systems that *work*. The gap between inherent and operational goals should be measured and overcome by evolving systems to allow users to "see, feel, touch, and taste" the system capability as it evolves, not by attempting to have HQ USAF direct perfection at the outset of, and on a day-to-day basis during the acquisition process.
—The funder/user/developer relationship must be modified to represent a customer/supplier relationship. Currently, the developer has two customers—a funder (HQ USAF), and a user (the operational command). Satisfaction of both is often mutually exclusive, with the developer caught in the middle. The user and funder should harmonize overall system goals, with the user and developer coupled in an sense of a customer/supplier relationship. In short, HQ USAF should avoid a detailed involvement in the design and acquisition of the capability as long as the user is satisfied with the customer/supplier relationship, and funding integrity is being maintained in the aggregate.

Observations

Throughout this paper, we have dwelled on the complexities of C^3 acquisition in terms of interrelated problems and/or symptoms. The question is, where do we go from here?

The United States is not deploying C^3 systems commensurate with its need, nor with its technological capability to produce these systems. To underscore the importance of C^3 systems, one needs only to examine World War II in light of recent revelations about the *Ultra* decoding device. A legitimate case can be made that the Allies won the war because of the superior application of information to the strategy and tactics of warfare. *Ultra*, combined with personnel, procedures, and equipment for disseminating vital information to theater and lower level commanders, demonstrated the absolute need for a working C^3 system. Herein lies the first of several essential changes in acquisition philosophy necessary to reduce C^3 acquisition problems.

The overall thrust of existing and foreseeable C^3 acquisitions must be changed from building *ideal* systems to building *workable* systems. Watson Watt said (in reference to British radar systems) ". . . the best design had to be rejected because it would never be achieved, and that the 'second best' would be achieved too late to be used by the armed forces when they needed it. The third best would be adequate and was available in time, and it was what won the Battle of Britain." A more contemporary expression of the same principle is, "If war comes, a good

system in the field is worth any number of ideal systems on the drawing board."

In short, we must lower our sights and work toward more realistic objectives, incrementally building as requirements evolve and are understood. System architectures must be structured to accommodate change.

Ideal system as a goal of a C³ acquisition is proper; attempts to direct perfection by continually changing short-range objectives of a C³ acquisition is unworkable. The objective of each C³ acquisition must be a working system, not an ideal system. All levels—Congress, the Joint Chiefs of Staff, service staffs, developers, and using commands must recognize this traumatic shift in philosophy. This does not portend a change in Department of Defense directives or Air Force regulations. Extant guidelines do not preclude this concept. However, we are mistaken to think that because a regulation does not preclude a given action, the action will be allowed. We need a major change in attitude, at all levels, from idealism to pragmatism. This is perhaps best expressed by paraphrasing D. K. Barton: "We should find the optimum solution and use it as a reference, but we should avoid attempts to build it."

This working-objective philosophy must be implemented by first modifying the traditional approach to the conceptual and validation phases of acquisition—by introducing prototypes in the form of joint developer/user operational testbeds. These prototypes must be used to force the concepts of C³ systems through "see, feel, taste, and touch" of the systems in response to operational environments.

Validation of detailed C³ system requirements by higher headquarters must be replaced by higher headquarters endorsement and approval of overall C³ system goals. This can be achieved by the identification of national information needs, and by controlling the allocation of funds against the shred-out of these needs by user or system application, not against detailed using-command system requirements.

We must, in our regulatory procedures, reject two more traditional acquisition constraints. We must adopt a "software first" approach. The traditional approach to standardizing hardware is not cost-effective. Witness WWMCCS: If the individual using commands cannot meet their operational requirements because of standardization, then nothing is achieved, nor is money saved, through standardization. Standardization and interoperability can be achieved through the proper complement of hardware, after the basic system software capabilities have been established and are working in a prototype-user-employed facility.

"Software first" also means optimizing based on total system cost. Hardware cannot be used as a basis for cost judgment and, just as important, it cannot be used as a basis for competition. If software works on an IBM machine and the

machine becomes obsolete, then buy a modern, compatible IBM machine, sole-source if necessary, and without competition. More generally, as software is proved to work in a developer/user prototype, and is ready for transfer to the operational evnironment, then hardware that is compatible with the software should be purchased. The software-first concept can be extended to the prototype environment by establishment of file constructs that will meet the information requirements, building the system in user facilities on a development machine as a demonstration, then contracting for hardware and conversion of the existing software.

Recommendations

—Establish a DOD-wide C^3 system architecture that lays out time-phased goals, not detailed requirements for subordinate C^3 elements to meet.
—The Air Force Systems Command should provide architectural assistance to its C^3 user command customers to aid in setting user-command goals. (The architecture responsibility is a user responsibility).
—The user command, with AFSC assistance, should be given responsibility for advocating programs and obtaining funds approval for C^3 system improvements.
—The user command should be given the authority and responsibility for the evolutionary and incremental improvement of its C^3 system towards HQ USAF approved goals. Overall funding should be based upon a "level-of-effort" improvement philosophy in contrast to infrequent massive changes in user capability, and should be provided to AFSC by HQ USAF against user-validated needs.
—The user command should "contract" directly with AFSC for analyzing needs, assessing state-of-the-art, planning improvements, acquiring contractual help, acquiring and installing equipment, and controlling the developments that support system evolution. An appropriate joint-management structure to support this close working relationship must be established. The user command must be required to go to AFSC for system development and equipment acquisition, consistent with the AFSC funding level approved by HQ USAF.
—The Air Force Systems Command should establish, with the user, C^3 system prototypes and appropriate test facilities to support the evolutionary system development/acquisition process (such as was done for SAGE through the establishment of the Evaluation Sage Sector, a full-up user, developer, support command prototype exercised with operational aircraft, people, and procedures).
—The Air Force Systems Command should establish prototype-centered advanced development (6.3) and engineering development (6.4) supporting programs to lend context to the development and application of the latest technology to user needs.

These recommendations, when taken together, portend a larger up-front investment of development dollars in C³ systems. This must be viewed as an investment in the future, because it provides the means to the desired end of reducing the expenditure of development dollars on C³ systems over the long haul by allowing us to field needed capabilities efficiently, within cost and on-schedule.

An Evolutionary C³ Systems Acquisition Strategy

In light of the foregoing recommendations, ESD has formulated and is applying to certain C³ systems, a variation of the traditional three-phase system acquisition approach. This variation achieves a desired capability in a series of development steps, in contrast to the traditional approach that accomplishes full-scale development in a single step. In the multistep approach, the first step provides a minimum capability. Program approval, funding, and contractor selection for this minimum capability are, in an overall sense, the same as for an all-up capability. Subsequent steps are described and budgeted on a representative basis, since firm definition will be based on the experience gained from the preceding step. Because of budgetary lead times, subsequent steps will be specified and negotiated within approved budgets. An "evolutionary" description is used instead of "incremental," because subsequent steps will call for both new work and modifications to old. The evolutionary approach allows for user participation as a member of the SPO team, where the user has an application software development capability derived from a requirement to operate and maintain application software systems. The user may develop the application software system himself, or with contractor support. The Electronic Systems Division retains overall responsibility for system development and acquisition. This approach has been employed in both ADCOM and TAC C³ systems with good results, after exercising significant problems with the more traditional approach.

The evolutionary approach is ideal for software-dependent systems where full definition of requirements before program initiation is impractical.

—It maximizes user involvement in both the formulation of system specifications and the development of the application software subsystem.

—Although budgeted on a representative basis, contracting of subsequent steps, which can be done by exercising options, is not accomplished until requirements and specifications are well in hand, including incentive and cost-sharing provisions.

—No fundamental changes are required to Air Force and Department of Defense regulations. However, a departure from the concept of full system definition and the acceptance and employment of budgets based on representative development packages require both DOD and congressional understanding.

—Independent IOTE is accomplished in such a manner that changes resulting from testing can be accommodated and contribute to the definition of requirements for succeeding steps.
—The evolutionary approach will shorten the acquisition cycle time by eliminating the requirement for an all-up system definition as a prerequisite for program approval and initiation.‖

191

Joint Service Software Policy and Standards

Lieutenant Colonel Casper H. Klucas, USAF
Major Larry A. Fry, USAF
Lieutenant Commander John W. Barnes, USN
Dr. Matthew Fisher

Over the past decade, the complexity of military systems has increased steadily because of the growing demands of modern warfare. These demands have not only been for increased performance, but also include requirements for more flexible and adaptable weapon systems. The prime instrument for providing this flexibility and adaptability has been the embedded computer which, because of its programmable nature, can be readily changed to meet varying mission needs. However, the labor-intensive nature of programming these computers has made software a major cost driver in the development and deployment of defense systems.

In response to the explosive application of computers in defense systems, the attendant spiraling software costs, and the many instances of late, over-budget, and unreliable software delivery, the Department of Defense started a number of initiatives in the mid-1970s aimed at providing better management of software development projects. Within the Office of the Under Secretary of Defense (OUSD), DOD Directive 5000.29 was developed and published. It established DOD policy for the acquisition and support of computer resources in defense systems, commonly referred to as "embedded computer resources." The services then developed their own policies for acquiring and supporting computer resources, which promulgated the DOD policy and, in most cases, added service-unique interpretations to that policy. The existence of service-unique policies also led to the development of acquisition and documentation standards oriented to the specific needs of each service.

Lieutenant Colonel Casper H. Klucas, USAF, is Chief of Computer Resources Division, Headquarters Air Force Logistics Command (AFLC), Wright-Patterson AFB, Ohio. He is responsible for establishing computer resources management and support policy for weapon systems assigned to AFLC. Lieutenant Colonel Klucas chairs the Joint Logistics Commanders' Computer Software Management (CSM) Subgroup responsible for developing the joint-service policy regulation.

Major Larry A. Fry, USAF, is the Deputy for Computer Resources Policy and Practices within Headquarters, Air Force Systems Command, Andrews Air Force Base, Md. He is responsible for software developing policy, standards, and procedures for the acquisition of mission-critical computer software. Major Fry is the Air Force Systems Command representative on the Joint Logistics Commanders' Computer Software Management Subgroup.

Lieutenant Commander John W. Barnes, USN, is a Software Engineer in the Computer Resources Division, Naval Electronics Systems Command (NAVELEXSYSCOM), Washington, D.C. His responsibilities include computer resource policy formulation and software engineering support to NAVELEXSYSCOM program managers (PMEs). Lieutenant Commander Barnes is the Naval Material Command's representative on the Joint Logistics Commanders' Computer Software Management Subgroup.

Dr. Matthew Fisher is chief of the Advanced Technology Branch in Navigation for the U.S. Army Avionics R&D Activity, Fort Monmouth, N.J. From 1975 to 1981, he was a senior staff engineer for software and battlefield automated systems in product assurance for the Army. He received B.S. and M.S. degrees in electrical engineering, and a Ph.D. degree, all from Drexel University. He is a member of the Joint Logistics Commanders' Subgroup for Computer Software Management that is working on computer resource standardization for the Department of Defense.

In January 1979, the Joint Logistics Commanders (JLC) Joint Policy Coordinating Group on Computer Resources Management (JPCG-CRM), recognized an opportunity for service standardization and chartered a subgroup to review policies, procedures, regulations, and standards related to computer software acquisition and support. The subgroup, called the Computer Software Management (CSM) Subgroup, planned and conducted a software workshop in April 1979 to examine the services' software acquisition guidelines, management procedures, and standardization efforts to determine if there was a basis for coordination and adoption of joint-service policy and standards. The workshop panels concluded that the services should develop common policy, development standards, and documentation standards instead of continuing to approach software development in a service-unique manner. It was felt that economics were certainly one reason for developing a joint approach, but not the only one. By developing a joint approach the best methods of each service could be adopted for all to use.

Perspective

The first task of the CSM Subgroup was to develop a joint-service regulation for computer resource development and support. Without service agreement on the general policy aspects of development and support, there would be no point in developing joint standards and data item descriptions for software. In this paper we intend to describe the draft joint service policy regulation. We emphasize that this does not represent an approved or coordinated position of the Joint Logistics Commanders.

Framework

The first step in developing policy for computer resource acquisition and support was to define where the software development process fits. It was obvious that emphasis was going to be on the software and, therefore, the software phasing was vital to the development of a software model.

The approach chosen was to lay out the software development process as it *typically* occurs during the system life cycle (Figure 1). The software development cycle was then broken down into six subphases: requirements analysis, preliminary design, detailed design, coding and unit testing, software integration and testing, and software performance testing.

The outline for the regulation was then developed (Figure 2), providing a chapter for each phase of the system acquisition life cycle, i.e., concept exploration, demonstration and validation, full-scale development, and production/deployment, within which policy was established for computer resources and

FIGURE 1
Software Development: Life-Cycle Perspective

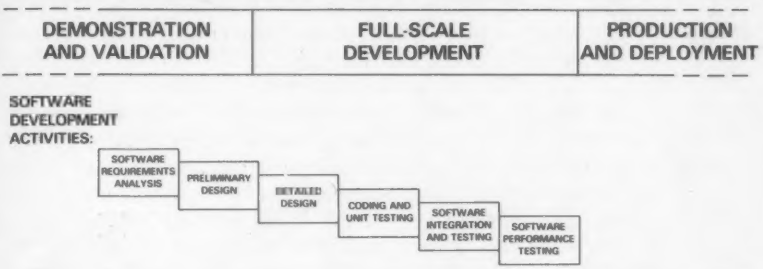


FIGURE 2
Organization

CHAPTER	SUBJECT
1	INTRODUCTION
2	COMPUTER RESOURCES: FRAMEWORK AND POLICY
3	CONCEPT EXPLORATION
4	DEMONSTRATION AND VALIDATION
5	FULL-SCALE DEVELOPMENT
6	PRODUCTION AND DEPLOYMENT
APPENDIX A	DEFINITIONS
APPENDIX B	COMPUTER RESOURCES LIFE-CYCLE MANAGEMENT PLAN
APPENDIX C	REFERENCES

highlighted in terms of activities, products, reviews, and baselines. Within full-scale development, a number of subphases occurred because of the typical concentration on software development during full-scale development. For each subphase, the policy was written to include activities, products, reviews, and baselines.

It should be pointed out that Figure 1 represents not only the system acquisition life cycle but, more importantly, where software development typically fits within the system life cycle. Because practically no program follows a typical path, the policy was written to recognize that software development could occur in any of the phases of the system life cycle. Therefore, the policy was written to require the addressing of software activities, reviews, products, and baselines no matter when software was developed. If, for example, software was actually being developed during the demonstration and validation phase, then the software policy in Chapter 5 of the regulation applied to that software development effort. Similarly, if software development occurs during the deployment phase (which is certainly the case when modifications occur), the policy within Chapter 5 applies to the software development or modification activities.

As part of the framework definition, the CSM Subgroup added two additional reviews within the software development cycle: the software specification review (SSR) and the test readiness review (TRR). The purpose of the SSR was to authenticate the allocated baseline for software prior to the conduct of preliminary design. The TRR was added to provide a review and status check prior to the conduct of formal software performance testing.

The following policy discussion describes computer software development and its relationship to the system acquisition life cycle.

Concept Exploration Phase

The concept exploration phase encompasses conceptual activities to define system-level operational and performance requirements, analyze development concepts, define interfaces, and develop initial planning documents. Similar conceptual activities are accomplished for computer resources, including initial computer software development cycle planning.

The concept exploration activities begin with the identification and validation of a mission need requiring a new capability. Once the need is validated, a program is initiated to explore alternative system concepts. Exploration of concepts involves refining proposed solutions or developing new concepts for satisfying the operational need. Exploratory activities include studies, trade-off analyses, experimental development, and concept evaluation. System engineering studies are conducted to explicitly address computer resources to the maximum extent

possible. The product from this phase consists of the initial system specification that documents total system performance requirements, including computer resource requirements, design constraints, and system interfaces. For major systems, a test and evaluation master plan (TEMP) is developed to provide an integrated plan of all testing for the system.

A system requirements review is conducted to evaluate allocation of mission requirements to system requirements, providing a basis for allocating the system requirements to hardware configuration items, or to computer software configuration items (CSCIs). Upon completion of the system requirements review and acceptance of the system specification by the development activity, the system specification may be entered into the functional baseline for the system.

Demonstration and Validation Phase

The objective of the demonstration and validation phase is to validate system requirements, allocate these requirements to configuration items (CIs), and to complete computer software development cycle planning before proceeding into the full-scale development phase.

System performance requirements are finalized and the functional performance requirements for each hardware and software configuration item are defined and validated. The functional interfaces and the necessary design constraints are also defined. Risk analysis is performed to eliminate or minimize the risks associated with achieving performance, cost, and schedule objectives. This analysis ensures conformance of planned computer resources with specified operational requirements.

Computer resources acquisition and life-cycle planning factors are identified and specific guidelines established to ensure that these factors are adequately considered in the acquistion planning process. The computer-resources acquisition strategy is finalized and resource requirements for the remainder of the system acquisition life cycle are determined. Planning is completed for procurement of computer resources during the full-scale development phase, giving particular attention to procurement of computer software.

The final system/segment specification, hardware configuration item specification, one or more software requirements specifications (SRSs), and the initial computer resource life-cycle management plan (CRLCMP) result from this phase. The CRLCMP is prepared as soon as possible to plan for acquisition during the full-scale development phase. The CRLCMP is updated as necessary throughout the system acquisition life cycle.

A system design review (SDR) is conducted to evaluate the system design and the allocation of system functions to individual hardware and software configu-

ration items. The system requirements specification for each software configuration item and hardware specification for each configuration item may be entered into an allocated baseline at the completion of system design review. If the system specification has not previously been authenticated, it is authenticated at the system design review and establishes the functional baseline for the system.

Full-Scale Development Phase

The full-scale development phase is the period during which the system, including computer hardware and software, is designed, developed, tested, and evaluated. This phase produces a system that closely approximates the production item, the documentation necessary to enter the production and deployment phase, and the test results that demonstrate that the system to be produced will meet the stated requirements.

Although described as sequential activities, the use of a top-down development approach may cause computer software development activities to proceed concurrently; different portions of the computer software may be developed in parallel, each proceeding sequentially through preliminary design, detailed design, coding and unit testing, software integration and testing, and software performance testing.

Requirements Analysis. Requirements analysis for computer software is accomplished primarily during the demonstration and validation phase. Requirements analysis may extend into the full-scale development phase during which computer software requirements are finalized. A complete set of computer functional requirements, including interface requirements, are established for the computer software configuration item. Requirements analysis consists of evaluating requirements for completeness, adequacy, testability, and understandability. This analysis continues throughout the computer system acquisition life cycle.

The major product of requirements analysis is a software requirements specification for the software configuration item. Interface requirements specifications (IRSs) are developed as required. A software specification review (SSR) is conducted to authenticate the system requirements specifications and interface requirements specifications. The purpose of this review is to determine whether the software requirements establish a satisfactory basis for initiating software design. At the completion of the software specifications review, the system and interface requirements specifications are entered into all allocated baseline.

Preliminary Design. The purpose of preliminary design is to translate the software requirements into a top-level design for each computer software configuration item. The design approach is selected and a modular computer software ar-

chitecture developed by considering various design alternatives and performing analyses and trade-off studies. The design defines mathematical models, the allocation of the functional requirements to components of the computer software, the relationship between these components, the external interfaces, and the data-base design. Software testing is concurrently planned with the development of the design.

The major products are a software top-level design document (STLDD) and a software test plan (STP) for each software configuration item. A preliminary design review (PDR) is conducted to review the basic design concept for each software configuration item. The purpose of this review is to evaluate the design and the degree to which it satisfies the software requirements before proceeding into detailed design. After successful completion of the preliminary design review, the top-level design document is entered into the developmental baseline, which is maintained under contractor control until final acceptance by the government as the product baseline. Concurrent with the preliminary design review, the software test plan is reviewed for approval by the government.

Detailed Design. The purpose of detailed design is to extend the level of detail of the software top-level design by identifying individual software units and by defining these units in sufficient detail for coding and unit testing. The detailed design is defined, including functional descriptions, logical flows, algorithms, formats, constraints, and the data base. Trade-off analysis of sizing and timing is performed to support development of the detailed design. Interfaces to other software configuration items, hardware configuration items, and data bases are designed.

The detailed design is documented by the software detailed design document (SDDD). An interface design document (IDD) and a data base design document (DBDD) may also be developed. Software test descriptions (STDs) are developed for each level of testing planned for the software configuration item. A critical design review (CDR) is conducted for each CSCI. The critical design review ensures that the detailed design solution, as given by the software detailed design document, satisfies requirements of the software requirements specifications and software top-level design document before initiating coding. After the completion of the critical design review, the detailed, the interface, and the data-base design documents are entered into a contractor's development baseline.

Coding and Unit Testing. Coding and unit testing are initiated after the successful accomplishment of each critical design review. Coding consists of translation of the detailed design into software units that will perform the specified functions. Unit testing is performed to check the computer software for correctness and to eliminate errors. After unit testing, software units are entered into the contractor's developmental baseline.

Software Integration and Testing. Tested software units are integrated for intermediate-level testing and CSCI-level testing during the software integration and testing subphase. Integration and testing normally begins upon completion of unit tests and proceeds through successively comprehensive levels of integration until the complete CSCI has undergone testing. Preliminary qualification testing (PQT) may be conducted to evaluate the computer software against requirements stated in the software requirements specification, and to provide visibility into the period between the critical design review and acceptance testing.

Software Performance Testing. Software performance testing is conducted to determine that the integrated software satisfies the requirements specified in the software requirements specification and applicable interface requirement specifications. Formal qualification testing (FQT) is conducted after software integration requirements specification. After successful formal qualification testing, both functional and physical configuration audits (FCA, PCA) are conducted to verify that all documentation is current and accurately reflects the software product.

A draft software project specification (SPS) is produced for the physical configuration audit and when authenticated, the final software product specification is produced. The functional and physical audits, respectively, ensure that each CSCI meets its computer software requirements and that its associated documentation is an accurate reflection of the computer software configuration item. A formal qualification review is conducted, at which the government officially certifies that the CSCI is fully qualified. The formal qualification review may be part of the functional configuration audit if that audit can demonstrate that the CSCI meets all system requirements. Otherwise, a separate functional qualification review is conducted after the functional configuration audit, normally upon completion of system integration and test. After successful completion of functional and physical configuration audits, the authenticated software product specification establishes the government's product baseline.

System Integration and Testing. System integration and testing validates that the complete system of hardware and computer software is properly integrated and satisfies system requirements. After successful acceptance testing, CSCIs and hardware CIs are integrated to form the system. The software is executed to ensure correct interfacing to the hardware components and satisfaction of system requirements. Documentation and software units in the product baseline are updated to correct deficiencies found in the software.

After the completion of system integration and testing, functional and physical configuration audits, and the formal qualification review are conducted as system-level assessments of the integration system, including a reassessment of the computer software from a system perspective. The product baseline is updated to reflect any software correlations.

Operational Testing and Evaluation (OT&E). For major system acquisition, OT&E is conducted before Milestone III to assess the system's operational effectiveness and operational suitability. Operational testing and evaluation may also be conducted for less-than-major systems, and is accomplished in an environment as operationally realistic as possible. Software modifications needed during the production and deployment phase may be identified as a result of the OT&E effort.

Production and Deployment Phase

The production and deployment phase consists of two major activities, production and deployment. In the normal system acquisition life cycle, these activities overlap. The production activity extends from production approval until the last system item is delivered and accepted. The deployment activity starts with delivery of the first operational unit and terminates when the system is removed from the operational inventory.

The purpose of production is to efficiently produce and deliver effective, supportable systems to users. Acquisition activities are completed before initiating production efforts. Provisions are developed for follow-on support contracts to maintain the currency of resources. Computer resources plans are updated as necessary for the remainder of the system acquisition life cycle.

Reviews are conducted as appropriate to reflect changes to the product baseline, which remains in effect and is updated to reflect software changes.

Deployment. The purpose of the deployment activity is to distribute the computer software to the system users, to maintain the software in response to evolving system requirements, and to correct deficiencies. During deployment, responsibility for managing the software transitions from the development activity to the Software Support Activity (SSA).

After the system is in operational use, changes to computer software may be necessary. For the purpose of controlling and validating software, these changes may be separated into two categories: system refinements to achieve program optimization, error correction, improved system performance, and technological advancements; and new requirements resulting from evolving system requirements or new applications. The extent of the software change dictates whether the change can be treated in the deployment environment or must undergo a redevelopment cycle with activities as identified in the computer software development cycle.

Updated documentation is produced as needed to reflect software modifications. Reviews are conducted as appropriate for the software modification.

Policy Implementation

A draft of the above-stated policy was distributed within the JLC commands in January 1982. Comments have been received and a revised regulation is being prepared. Coordination of the updated policy is scheduled to begin this fall with a sign-off by early 1983. Adoption of a joint policy by the JLCs will provide the necessary framework for implementing the software development and documentation standards discussed below.

Development Standards

Like the policy area, software development standards were developed to address the "unique" needs of each service with little coupling to standards in other areas. The emphasis within the software development standards arena was to provide for and require a disciplined approach to software development. However, the discipline that evolved was service unique and, in certain instances, did not provide a complete and proper coupling with the configuration management (CM) and quality assurance (QA) world. The improper coupling and existence of unique standards for each service proved to be costly to implement within industry. Unique contractor management control systems had to be established to satisfy each service's requirements.

The April 1979 software workshop identified a need to develop a set of tri-service standards so that each service would present "a common face" to the industrial community. The CSM subgroup defined a program to develop a military standard for software development, called MIL-STD-SDS, and a set of changes to MIL-STD-490, 483, and 1521. In addition, the CSM program included a task to develop software documentation standards that identified the types of documents needed to support mission-critical software development and operational use. A set of standards and data item descriptions has been developed by the CSM and was distributed to the services in June 1982 for formal review and comment. Copies were also provided for informal industry association comments. Comments will be incorporated into a revised set of development standards and data items for DOD and industry-wide formal coordination in early 1983.

Summary

The JLC activities in computer software management are well under way and represent a significant effort on the part of the services to adopt common policy and standards for software development. The potential for cost savings is significant. No longer will industry be required to maintain muliple-management control systems. By providing a more uniform environment for software develop-

ment, joint-service programs should run more smoothly. Although there is still a long road ahead in the policy and standards coordination and revision process, the end is in view. ‖

Software Needs Top-Down Management

202

David R. Brown, Jr.

Digital computers, as tools for serving an increasing variety of public and private interests, began to emerge with both scientific and economic effectiveness during the late 1940s. The evolving history of digital computers is marked by expanding horizons of applications and capabilities. Remarkable advances have been achieved in computer hardware and programming technology—computing speed, computer size, main and mass memory capabilities, programmability, including higher-level programming languages, and available interactive techniques. Many things on the material and theoretical forefronts of progress could not exist if modern digital computers had not been advanced to their present capabilities. The belief is, however, that a review of the use of computers, as general-purpose computing machines, *per se* and as incorporated in systems utilizing digital control, during these first 30-plus years will reveal that, in general, computer hardware technology and computing technology (i.e., data processing and programming methodologies) have far outpaced the equally needed development of software management practices. This is especially true for the increasing number of systems that incorporate computers, but whose basic purpose is not strictly computational.

Since their first use, digital computers have been conceptually associated with automaticity (e.g., automatic data processing). As a result, there often has been insufficient management attention given to the essential human aspects of software changeability and configuration management, system operator training, transfer of support responsibility, testability, and the general interrelationship between software and the functional capabilities of the systems that incorporate digital computers.

Without appropriate management practices, potential cost effectiveness in an overwhelming number of applications will not be realized, and the basic purpose of the system will be jeopardized.

There appears to be a misconception among managers of systems incorporating digital computers that the chief personal qualifications needed concerning software for successful management of the system acquisition and utilization is to become familiar with programming technology, e.g., programming languages, structured programming, compilers, etc. Notwithstanding the importance of programming technology to software, in reality it is more important that manage-

---

David R. Brown, Jr., recently retired from Civil Service, was a program manager of digital weapons systems at the Naval Surface Weapons Center, Dahlgren, Va., with emphasis on software management, as these systems have evolved over the past 20 years—the Navy's strategic missiles, guided missiles, guns, digital intelligence systems, and more recently, the cruise missile systems. He also was chairman of the Operational Software Panel of the Navy Laboratory Computing Committee. Mr. Brown holds engineering degrees from the University of Virginia.

ment actions stem from an understanding, based on required system performance, of what is to be programmed, and why—that is, the software *content*, rather than the *process* of how it is programmed.[1] The belief is that the level of system effectiveness equated directly to the degree of understanding by managers of the dependence of the system performance on the content of the system software. Normally, there are two fairly distinct, but never independent, categories of technical personnel required to provide the system software. These two categories, as noted for the management understanding, again refer to software *content* and *process*. That is, there is a need for analysts who formulate the algorithms (the content) to be programmed, and for programmers who can bring to bear the most effective programming technology (process). Ideally, there is a strong working interface between the personnel in these two categories. The working interface should permit the programmer, already the process specialist, to become knowledgeable about the software content, through system experience and association with the analysts. Rarely does one person serve both roles. Unless the programmers have otherwise achieved this familiarity through, say, prior experience, managers must take appropriate and timely action.

Software management practices vary among individual systems. There are many cases of less-than-adequate software management—a disparity between software management practices and both computer hardware management and computing technology. However, there are some exemplary cases to the contrary, where significant software management attention has occurred. These latter cases are usually a result not of change but of individual system planning. The chief ingredients of these successes are readily identifiable and can serve as guides for current and future systems applications of digital computers. Top-down software management practices, addressed later, need to be identified, understood, and practiced. Technological advances alone will neither solve software management problems nor make software management unnecessary.

Foremost among matters that military systems managers must understand and base their actions upon concerning the software of their systems, are the following:

—The system functional capabilities and the software are inextricably compounded.

—Software is the joint product of analysts and programmers.

—Continuity of software knowledge for a given system must be sustained for both the analysts and programmers.

---

1. The author acknowledges Dr. Richard Fairley, Computer Science Department, Colorado State University, as the source of the terms "process" and "content" in regard to software. These arose in conversation with Dr. Fairley in July 1980 at the Naval Surface Weapons Center.

—The length of time to achieve the necessary know-how is substantial and is extremely sensitive to management action.
—Software design rarely becomes fixed, but evolves with system usage requirements.

I will address these and other pertinent matters in subsequent sections.

Computer Systems and Digital Systems

Before examining what may be the major causes of software management problems, it is helpful to review the characteristics that distinguish computer systems from digital systems that incorporate computers. Strictly speaking, a computer system is the now-familiar, general-purpose computer system. The second is a special-purpose digital system, the software of whose computers serves the basic system functions. The epithet "embedded computer system" is sometimes used to denote this second type of computer application. In this paper, it is the second type that is chiefly addressed and will be referred to simply as a "digital system" (a system whose capabilities depend on digital control).

The General-Purpose Computer System

General-purpose computers have about a 30-year history and are now in their sixth or seventh generation—from relays and vacuum tubes to transistors and beyond. There is no doubt that pertinent new technologies (e.g., Josephson Junctions) will continue to unfold. It is common knowledge that the general-purpose systems have had far-reaching influence in the technical and commercial fields. They are used for an extensive array of computational applications, broadly categorized as data processing, engineering analysis and design, and mathematical and scientific research. The computers of most general-purpose facilities usually are not associated with just a single user. The computer applications are realized through the computer programs written as required, usually by the professionally trained users. For the general-purpose system, the computer hardware facility manager typically is not the same as the applications computer programming manager; the two matters are usually addressed jointly only when mutual scheduling matters are addressed, or when the facility specifications are defined. Additionally, the diverse applications are the separate concerns of the many users, who are, for the most part, organizationally independent. Albeit there is a working interface; whether there is or is not a unified, single management for the facility and the computer programs is not of critical importance. It should not be assumed, however, that this is the case for digital systems.

FIGURE 1
Digital System Software

Software is a primary system component that enables or supports realization of:

—Basic System Purposes and Functions
—System Hardware Design Capabilities
—System Operating Procedures
—Trained Personnel Abilities
—Pertinent User Command and National Doctrine

Digital Systems

Digital systems now have about a 20-year history, with an upsurge in the past 5 or more years. These special-purpose systems are distinguished by the incorporation of at least one digital computer among the system hardware components. A digital system is one that is dependent on digital control. It has specific functions such as controlling a power plant operation or other large-scale industrial process, launching a satellite or missile, or navigating a ship or aircraft. The software of a military digital system is expressly designed and used to serve the basic functions of the overall system; it should be technically coordinated within the same perspective as the total system specification. The software of a digital system is logically, therefore, a matter of system-level concern, uniquely associated with the bedrock purpose of the system throughout its entire life (Figure 1). Software management for a digital system should be constituted, enacted, and practiced accordingly. Moreover, the software management requirements for a military digital system are clearly distinct from those of a general-purpose system. A digital system has intimately related functions spanning its various components and user interactions. In comparison, the general-purpose system largely serves a kaleidoscope of unrelated computational needs. Unfortunately, the evolving practice of software management for digital systems has, in many cases, not followed the logic that software is of prime concern at the system level. Whereas programming for a general-purpose computer system is frequently a matter more associated with *how* to program a specified set of procedures or given set of algorithms, the preparation of software for a digital system is always a matter of *what* to program, as well. That is the essence of the critical requirement for top-down management for military digital systems.

For digital systems, it is essential that military system managers provide for the preparation of a software development plan in the formative stages of system development. Only through such a plan can managers identify, and enable to be realized, all the critical milestones and interfaces to be encountered in the development stages. This permits the analysts, programmers, and validation and verification (V&V) personnel to work toward a single, technically coordinated schedule. Otherwise, sufficient time for development events cannot be allocated.

In the sections that follow, the military digital systems management concerns, which are believed to be of major importance to software, are discussed directly or indirectly, individually or in combination. Briefly, these matters are the following:

—Management perspective—software content and programming process
—Evolutionary system requirements
—Continuity of specific software knowledge
—Coordination of hardware and software design
—Centralization of software responsibility for a system
—Readiness preparation for software support

Probable Causes of Software Management Problems

THE SOFTWARE MISCONCEPTIONS

Overt evidence of software management shortcomings is that too often the fundamental matter of digital-system software management is not adequately addressed by military system managers. There are numerous and costly myths concerning the nature of software (see Figure 2). Appropriate lines of software responsibility often are not established and sustained. For whatever deep-rooted reasons, there are misconceptions about the nature of the matter, resulting in ill-founding of software management practices and support responsibilities. These misconceptions are evidenced frequently in the following three ways:

—Software is erroneously believed to be solely related to the computer and is simply a matter of programming.
—Computing technology speciously appears to hold the solution of software management problems.
—Software management, even when recognized as a requirement, is fallaciously thought to be achievable by paralleling the more well-known hardware management practices.

Often misled by the above mistakes of logic, attempts to overcome the software management problems have sometimes fallen short. Let's examine the above three misconceptions.

FIGURE 2
Costly Software Myths (Commonly Held by System Managers)

—Software is needed only to make the computer work and reflects no system function other than the computer operation
—Complete understanding of all the computer programs in the software package for a system is easily and rapidly achievable
—Technology that enables design and proof of error-free software is fully established
—Testing will ensure detection of all software errors
—The computer won't run, or will otherwise indicate if any erroneous software change is made
—On-the-spot software changes, without the normal testing, are acceptable risks
—Inherent flexibility of software obviates change control and central responsibility
—A software maintenance agent is needed only to perform on-call software repairs at the user site

Software erroneously associated only with the computer: It seems to be an almost irresistible urge to associate the software for a digital system strictly with the computer (or computers) incorporated (or embedded) in that system. Many of the critical factors involved in software are, of course, computer matters—word length, processing time, and main-memory size, to name a few. From the system perspective, however, these factors alone, important as they are, are not appropriate measures of the system performance. System software has to be associated with the total system, not just the computer. What is to be programmed is certainly a system-level subject. Software typically interacts with every system component, and every system function. Its design depends on the total system purpose and must accommodate every usage of the system, operable under any specified condition. The software also is the principal means by which most of the operator-machine interfaces are realized. Management of software for a system should be established in the earliest phases of system development and sustained without interruption for the entire life of the system (see Figure 3).

The software package for a digital system comprises all the computer programs that execute on the incorporated computers of that system. This set consists of the basic applications programs, which are associated explicitly with the prime system functions, the system and component operability and readiness

FIGURE 3
Rudiments of Software Development and Acquisition Management

Management action must provide technical coordination, based on the development and support plans, and ensure concurrent preparation for system usage:

—Continuous technical, in-depth know-how of the software (content and process)
—Dedicated software support facility
—Configuration Control Boards (Change Control Boards) which are user responsive with participant representation by system management, software development and support activities, training centers, and user activities
—Data Base Management Systems (as required)
—Training Center courses, instructors, and facilities

programs, the hardware fault diagnostic programs, the support software such as compilers, assemblers and other utilities, documentation, V&V software and system test-equipment programs (see Figure 4). With that spectrum of significance to the total system and its users, just citing names of the software package categories clearly shows that software is not just a matter of computer programming. Yet, a system organizational focal point for the software is often not established and coherently sustained, with the corresponding result that configuration management practices become diffused and ineffective.

Computing technology is erroneously believed to preclude or diminish the requirement for software management: This misconception is manifested chiefly in matters related to software testability and learnability. Both qualities are dependent to some extent on the software package size and system functional complexity, among other factors. It is not unusual, for example, for a digital system to require software encompassing half a million computer instructions, sometimes representing hundreds of operator/machine functional interfaces. The voluminous task of designing this much software represents the orchestrated effort of many system design personnel. Even such superficial statistics signal the difficulties encountered in software testing and learning.

Testability refers to the ability to test software instructions individually and collectively. It is essential that software be tested as originally designed and whenever design changes are made. Design changes are required at any time in the system lifetime for added capabilities and for the correction of instruction er-

FIGURE 4
Digital System Software Package

- **—Basic System Software**
- **—User Test and Maintenance Software**
  - **Operability and Readiness Tests**
  - **Hardware Diagnostics**
  - **Peripheral Equipment Tests**
- **—Support Software**
  - **Compilers, Assemblers, Utilities**
  - **Simulation and Certification Software**
- **—Test Equipment Software**
- **—Documentation**
- **—V&V Software**
- **—Data Base Software**
- **—Training Software**

rors. Each instruction is designed to perform a given step in the overall process. Representing purely the programming, the instruction design could be in error. From the overall system function or algorithmic standpoint, however, the given process step or operation could also be incorrect because it could have been erroneously formulated or specified. Often, the actual computer code or subset of instructions to achieve a given program step is not unique; similarly, if this subset is in error, the effect(s) of the error is (are) not unique. It requires no stretch of the imagination to recognize the truth of the matter: Designing a suitable test to determine if the instructions, individually and as a network, are correct (or if incorrect, how?) can be more difficult and uncertain than the original design. This same situation also pertains to any subsequent modification of the instructions. The extensive software V&V efforts that parallel (by separate teams) the original software development must be sustained throughout the system usage to meet requirements for design changes and follow-on developments, representing the overall system evolution in general.

There is no known methodology for achieving software perfection. A computer program could be error free, but there is no way of knowing this is the case. Before issuance for use, a computer program or software package could pass every test to which it has been subjected, be certified as "debugged" and ready for use, and still contain crucial errors, whose effects, hopefully diagnosable, will sooner or later emerge.

Some research and development sectors in computing technology have long addressed the so-called "error free" programming problem. Some progress has been made for limited conditions. There is no assurance that such techniques could be developed to be effective at the levels of program volume and complexity characteristic of typical digital systems. Very likely, these techniques would not be of broad, general applicability. Rather, they would have to be specifically worked out for a particular, fixed set of computer instructions. Rarely has the software package of a digital system settled down to become a fixed set of instructions. Certainly it has not happened for any major military digital system. (The ongoing emphasis is that one of the critical measures of the quality of software management is the effectiveness of the control of the change process.) Moreover, the discovery and identification of instruction errors is largely not a deterministic matter, but rather a probabilistic one, no matter how well the tests are designed and performed. Allocation of time in system-development schedules for software design and checkout contains risk; certainly the shorter the period the greater the risks. This is the crux of the management problem for software testability.

In the context used here, learnability refers to the system personnel-readiness process for software support, as opposed to operator preparation for system usage. Learnability associates directly with how software analysts, engineers, and programmers become sufficiently familiar with the software package to do the following: design and implement modifications to the software; create functional extensions and additions to the existing software; test and analyze software at any stage of its existence; sustain configuration management; effect documentation; and carry out training and user liaison. The software team must be proficient in all of these at all times in the life of a digital system.

The time for system personnel to achieve the required status of this demanding capability depends directly on the experience the software analysts, engineers, and the programmers have had with the software package. The length of time can vary from several months to 2–3 years, even for the same system (see Figure 5). On the other hand, the level of effort (i.e., number of personnel) required to achieve this capability is more directly related to the size and complexity of the particular software package. The combination of level of effort and length of time basically determines the cost of support preparation. For software, at least for its present and foreseeable state of technology, there is only slight interdependence between the required length of time and level of effort to satisfy the learning curve. A scarcity of available time is not balanced by a plethora of effort.

The necessary level of effort for support preparation for a given system determines the preparation cost per unit of time. The length of time required, however, can, in most cases, be controlled by management actions. These actions must have been planned and initiated in the early phases of system development.

FIGURE 5
Digital System Software Support Responsibility

Learning Curve Dependence of Support Team Personnel on Developer Accessibility

Software Support Team	Team Preparation Time (months)
I. Development Team Follow-on	0 - 6
II. Separate from Developer	
A. With Access to Developer	
1. Development Participation	5 - 9
2. Development Monitor	8 - 12
3. All Specified Documentation	12 - 15
B. Without Access to Developer	
1. All Specified Documentation	18 - 24
2. Program Listings, Partial Documentation	24 - 36

Because of the enormous impact on system cost, the length-of-time factor needs to be examined more fully. I re-emphasize that the length of time is inversely proportional to the degree of familiarity that the software analysts, engineers, and programmers have with a system's software package. It may be true that no personnel are more familiar with the software for a digital system than those who initially developed the software, especially during the latter stages of development. If the software support team required for the usage phase of the system, sometimes referred to as the "life cycle," could be mustered from among the members of the development team, the resulting group would need a minimal length of time to be prepared for the support responsibility. This assumes that there is no significant gap in time between the end of the development phase and the beginning of support requirement. The necessity for that assumption is because if it is not directly and actively applied, software knowledge, once obtained, is highly perishable. It is of the utmost importance to avoid discontinuity, at any time in the life of a system, of knowledge of the software package, in part or in whole. The risk is identical to having spent insufficient time on the learning curve.

The primary preparation objective is to have the software support team "fully up" on the learning curve by the time that the demands of support responsibility begin. If these personnel are assembled from the development team, the learning curve requirements will be satisfied in minimum time with essentially no addi-

tional effort. Clearly, this condition represents the least cost of preparation. If the support team is to be assembled with personnel who have not participated in the development, the length of time and preparation cost depends on a gamut of possibilities directly resulting from actions management has taken, either by plan or otherwise. Essentially, the possibilities reflect the degree of accessibility that has, by management action, been accorded the support personnel to gain knowledge of the software package. For example, suppose that the software responsibility is to be transferred to a support activity whose personnel are assigned in time to monitor the development activities, with full technical communication with members of the development team; given also that the software is sufficiently documented, the preparation time for this case typically can be less than 1 year. As the next example, if the support personnel commence their preparation period subsequent to the development period, the starting time again being just a matter of management planning and action, the length of time needed will depend on how well the development team has documented the software, and the post-development accessibility of the development personnel for technical communication. If the documentation is incomplete or of poor quality, and if the software listings are not representative of the actual programs (this usually results from uncontrolled changes of the software, when configuration management practices are either unwittingly or knowingly ingored), and if the developers are no longer accessible, the preparation time can easily be 2–3 years. These examples are intended to point out the potentially vast cost savings that can be realized through proper management practices. These figures cited for length of preparation time represent real experiences with a broad variety of digital systems. These figures also assume that the support team has access to a system facility on which the actual software can be exercised. To summarize support preparation points, no reasonable projection of the length of time required for support preparation can be made unless attention has been given to the documentation and to preparation planning and scheduling that enhance the developer accessibility. The cost savings are in direct proportion to the results of these management practices.

Software management practices fallaciously assumed to be equivalent to hardware management practices: In what, perhaps, is considered to be the typical long-term process of the hardware for a new product or system being created and utilized, the people who design or develop the product or system hardware are usually not the same ones providing maintenance and support during the usage life. In terms of personnel, organization, function, and responsibility, the concept of a developer and a maintainer, as separate entities or agents, is commonplace. Sometimes the breakdown of function or responsibility is more diverse. For example, there are, for some military systems, designated activities to fulfill the

roles of design agent, support agent, maintenance agent, and in-service engineering agent. For digital systems, the widespread, but not universal, view appears to be that the hardware and software developments should be a combined effort by one activity. Certainly, the two should be technically coordinated by the system manager, especially for military systems. Actually, it can be more efficient and cost effective to have separate teams, even under separate but mutually drafted and managed contracts, to develop the hardware and software. One of the chief reasons for this is that the needed software management focal point, and visibility for the software development events, plus the benefits of checks and balances, are more readily achieved. System managers should strive to ensure software visibility and technical coherence. Their objective should be to have a focal point of software responsibility within the system management structure, rather than a dispersed responsibility as under the conditions cited above, where a series of agents are institutionalized, each with partial, and at times, confused responsibility. The debilitating dispersal of responsibility creates an unnatural and tenuous fabrication of functional support. Configuration management thereby becomes diffused, ineffective, and about impossible to implement.

The root causes of this misunderstanding are believed to lie in the often unrecognized, or erroneously perceived, fundamental differences between hardware and software. Perhaps most significant is the concept of maintainability. The widespread expectations are that hardware with normal usage will degrade or "wear out," and will require repair and parts replacement; hence, the terms "mean time between failure" (MTBF) and "mean time to repair" (MTR). But these terms, unfortunately, are not used in reference to software, because there is no concept of a computer program "wearing out"—it will change only through instruction design change or input modification, hopefully deliberate and correct. In terms of a modern digital system, the post-development hardware maintenance requirements are usually well understood, and the procedures, personnel, maintenance training, facilities, and parts inventories are planned and provided to support these requirements. Because a computer program does not physically "wear out," the means by which post-development requirements for software support are met are, by nature, fundamentally different that those for hardware. Any changes to software is actually a design change. The support procedures should be implemented accordingly. The establishment of distinct management practices that are actually required for software have not progressed to the same levels of workability as those classically achieved for hardware. The technical coherence and consistency needed for software support (see Figure 6) are best realized through central responsibility within the systems management structure for the entire software package for a military digital system.

FIGURE 6
Software Technical Activity Requirements During Digital System Usage Life

- —New System Capabilities (functions, procedures, hardware)
- —Configuration Control and Documentation
- —System Coordination Effort
  - User Command Doctrine and Procedures
  - User Activity Interface and Support
  - Training Center Interface and Support
  - User Activity Exercises
  - New User Activity Installation Tests, Overhaul Tests
- —Data Base Changes, Additions
- —Software Testing and Error Resolution

Paralleling the misconception of maintainability is the equally important one of changeability. Associated with the usage life of operational systems are the familiar terms "in-service engineering change," "alteration," and "field change." For the military system hardware, these changes usually refer to modifications made subsequent to the development period. These hardware changes are normally designed and prepared by the system design agent. Typically, the corresponding configuration management and control are performed by the system support agent; distribution and test planning are performed by the system maintenance agent; and on-site installation and test monitoring are accomplished by the in-service engineering agent. What is to be emphasized is that the technical substance of the design changes is still a responsibility at the development design level. In this sense the changes are not strictly "field" changes because they are not designed and implemented independently by the users of the system. For software, any change is a design change, but because of the relative ease of altering a computer program, changes are sometimes made in the field, by users and support activities that have not been designed by the design agent. Even though such changes may subsequently be approved and documented by a configuration control process (hopefully established) with specified testing and design refinements, the temporary risk can be perilously high. It is more effective, in function and cost, to have all software design changes made by the central support agent. Such a practice is not intended to suppress recommendations for change from the users, or from any system-associated source. Such user recommendations, reflecting military system operational experiences, are vital to the system support

and follow-on development process. The corresponding software design changes should, however, always be developed and implemented (or at least issued for implementation) by the agent having the central responsibility for the software.

Computers are so conceptually associated with automaticity, unfortunately, that often the operator training needs, as well as the software support team preparation, are inadequately addressed. Frequently, the user training courses and facilities are considered as items to be included under the usually hardware oriented, logistics support plan for a system. Sometimes managers, erroneously thinking that there is no need to ever change a computer program, relegate the software for a system to items of the logistics support plan, believing that program distribution is all that need be considered. Planning these preparations is essential, but it is the timing of the preparation and the working relationships that have the greatest impact on the training cost effectiveness for the entire life of the system. The training courses should be designed during the system development period. They should be designed having full accessibility to the software developers, i.e., personnel, facilities, and documentation.

The training courses should be tested rigorously by actual trial runs. A comprehensive working relationship should be established between military system training centers and the software development and support activity as early as possible in the system development period. This relationship should be sustained throughout the life of the system. Training courses must be updated in a timely manner whenever software changes are implemented by the support activity. Feedback from the training centers can be extremely valuable information concerning the software performance qualities. Training centers should be fully represented as members of the software change control boards. The operator-machine interfaces, the vast potential of the interactive capabilities of digital systems, can be exhaustively planned and designed, but can never be fully realized if the user training is not correspondingly effective.

Summary of Digital System Management Issues

For convenience, a summary of military digital system management issues, which are believed to be of major importance to software and which have been discussed in previous sections, is given in Figure 7. The list gives the subject and brief pertinent substance for each issue. These matters are referred to as issues because there is non-uniform practice concerning them among known digital systems.

FIGURE 7
Military Digital System Management Issues of Major Importance to Software

—Management Perspective
Must encompass:
- **Software content (system functions) as well as process (programming technology)**

—Evolutionary System Requirements
- **Demands continuity of specific software knowledge (content and process)**

—System Development and Support
- **Requires coordinated hardware and software effort by system management**
- **Separate activities for hardware and software efforts sometimes appropriate**

—Software Development and Support Responsibilities
- **System life retention of responsibility usually more effective and economical than transition to a support activity**
- **One activity responsible for the software product, with multi-activity change proposal**
- **Changes designed, incorporated, and introduced only by the software responsible activity**

—Software Support Readiness Preparation
- **Cost and time extremely sensitive to management action**

Epilogue

THE ACADEMIC PERSPECTIVE

Because software at the technical level is the joint product of analysts and programmers, there is typically no single integrated academic preparation for software personnel. The academic backgrounds of both analysts and programmers can represent a broad spectrum of scientific and technical fields. Moreover, the programmers have knowledge of programming technology, usually gained by a combination of experience and academic training in specialized technical fields concerning computer engineering and computer science. The programmer is more computer-oriented than is the typical analyst. The programming knowledge, that is, the software process knowledge, that the academic world provides is applicable to many software contents. In the academic world, however, rarely are

both the processes (programming technology) and content (the applied science or engineering that results in a algorithmic product of the analyst) taught in a unified curriculum. A totally separate curriculum that would cover all matters pertaining to software would be conceptually repetitious of many technical disciplines, and would overlap various fields of applied science and mathematics. The academic world provides the knowledge of the software *ingredients*, for both process and content, but the individual digital systems must provide their specific *recipes* for the software product. People emerging from academic preparation and entering the world of application for digital systems should have a reasonable feel for whether they want to specialize in programming or analysis. Hence, they choose accordingly, usually early in their careers.

The above reflections on "content and process" serve to emphasize the requirement for system managers to recognize the two basic categories of personnel resources involved at the technical level of software design. Neither category alone can support the required software responsibility demanded by modern digital systems. The fact that software is the joint technical product of analysts and programmers profoundly influences the resource requirements that managers of digital systems must accommodate. It affects all phases of the system acquisition and usage. The complex human factors attendant to the resource requirements are often the pacing items along the critical path of digital system design and usage.

THE MILITARY PERSPECTIVE

The risk of partial or complete failure of a digital system because of erroneous software, the configuration which, for reasons discussed below, has become unknown to the software design agent, is a potential reality that must be addressed by systems managers—before the chances of it happening become irrepressible. Especially for deployed military systems, the risk should be considered unacceptable. Certainly, if under combat conditions, for example, a whole weapon system were disabled because of insufficiently tested software, then incalculable losses could result. If such conditions exist because of practices sanctioned by system management, then it is a blatant and reprehensible lack of the rigorous configuration control that should be a primary and fundamental requirement of military systems design and usage. It should be a basic responsibility of system management to deliberately take action to avoid having untested software in deployed military systems. Unfortunately, the management record is not clean; lessons have not been learned, even when systems malfunction or missiles go astray for reasons of untested software. One thing, fortunately, is known about the cases where the hazards have materialized. The potentially perilous risks are created because system management did not establish a central respon-

sibility for software within the system management structure. There are examples of reports from military commanders that describe this situation, but the conclusions are ones of resignation and resorting to a bad practice rather than ones that lead to appropriate resolution. The reports tell of unannounced (i.e., undocumented) deliveries of "revised" or "modified" software to the system users from, or by, several of the system participating agencies. The user (or anyone else, for that matter) has no way of knowing if these changes are compatible. Sometimes the user doesn't even know what purpose these changes are intended to serve. The software is plunged into a state of uncertainty, and the system operating procedures are often unpredictably disturbed. Overwhelmed by a sense of hazard and unacceptable risk, the military commander tacitly assumes that the system management either cannot or will not initiate support practices that are appropriate. The result is that the users, operators of the deployed military systems, want to make software changes themselves as deemed necessary to overcome software errors or deficiencies. The point is not that the user, or any other system participant, is not sensitive to the needs of the system or is intentionally irresponsible. The point is that the means of adequately testing the software changes—which are design changes, not repairs or corrections—do not exist at user sites; nor do they exist at any activity other than the design agent. Only the activity having central responsibility for software should perform the design-agent function, because only that activity has the necessary means to test the software design change. No design change can be assumed to be acceptable without testing. The testing typically associated with software quality assurance, and with software validation and verification, is extensive for major systems, requiring continuous independent effort. For military systems, certainly, these efforts should be established and sustained as working parts of the system support. It is essential because there is no perfect means of testing software. If management permits, wittingly or unwittingly, untested and uncontrolled software changes to be made, the fundamentally important meaning of these testing essentials, in terms of military effectiveness and the safety of personnel and resources, is being ignored. There is no evident reason for such irresponsibility through inappropriate management practice. Technology does not offer impunity from such practices. To subject the national interests to such potentially perilous risks cannot be justified. ||

*U.S. GOVERNMENT PRINTING OFFICE : 1982 0-393-897/287

53

13

U·M·I

THIS JOURNAL CEASED PUBLICATION WITH
VOLUME 5, ISSUE 4 (AUTUMN 1982).